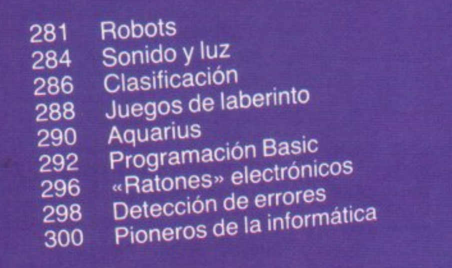


mi computer

15

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR



281 Robots
284 Sonido y luz
286 Clasificación
288 Juegos de laberinto
290 Aquarius
292 Programación Basic
296 «Ratones» electrónicos
298 Detección de errores
300 Pioneros de la informática





MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona, y comercializado en exclusiva por Distribuidora Olimpia, S.A., Barcelona

Volumen II - Fascículo 15

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Asesor técnico: Jesús Nebra

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti, A. Cuevas
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, Barcelona-8
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador,* se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-85822-90-0 (tomo 2)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 258404
Impreso en España - Printed in Spain - Abril 1984

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, Madrid-34.
Distribuye para Argentina: Viscontea Distribuidora, S.C.A., La Rioja 1134/56, Buenos Aires.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio Blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Ferrenquín a Cruz de Candelaria, 178, Caracas, y todas sus sucursales en el interior del país.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 16 690 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 087 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 3371872 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Distribuidora Olimpia (Paseo de Gracia, 88, 5.°, Barcelona-8), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Distribuidora Olimpia, en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.



Su fiel servidor

En la actualidad los robots industriales pueden reconocer objetos y aprender nuevas tareas imitando las acciones humanas

La palabra "robot" (del checo *robota*: trabajo) fue acuñada en 1920 por el escritor checo Karel Čapek en su obra teatral *R.U.R. (Rossum's universal robots: Los robots universales de Rossum)* para denominar a un androide creado por un científico y capaz de llevar a cabo trabajos realizados tradicionalmente por un hombre. El término fue acogido con gran entusiasmo por los escritores de ciencia-ficción. A pesar de los numerosos relatos novelescos que narran los poderes de los robots, éstos no son más que una extensión electromecánica del ordenador, con todas las limitaciones e imperfecciones propias de estas máquinas.

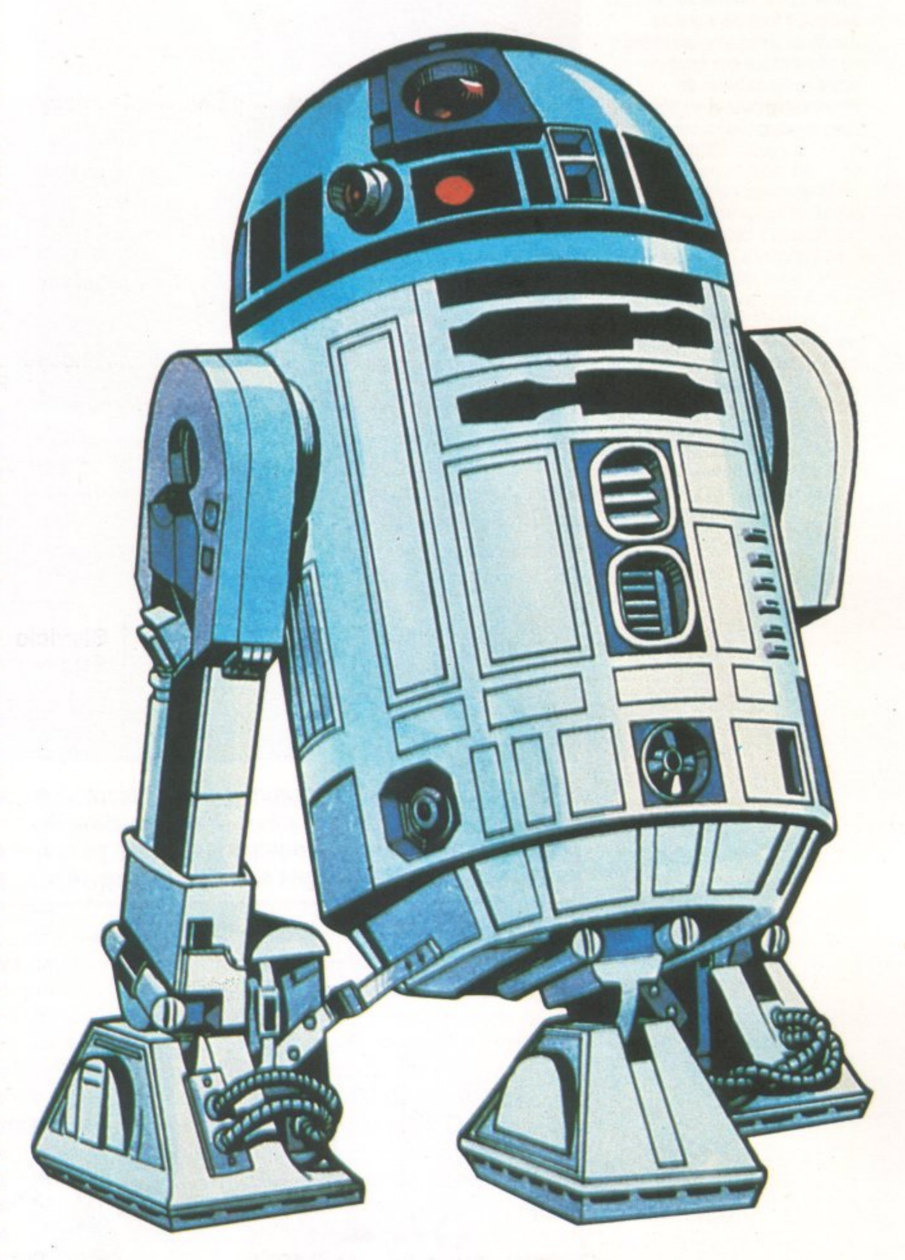
Sus orígenes se remontan a los talleres de maquinaria de los años cincuenta, en los que se aplicó por primera vez la teoría del control numérico a las máquinas-herramienta. Estos primeros esfuerzos fueron, previsiblemente, muy elementales: máquinas controladas por cinta de papel de cinco agujeros (del tipo de las que utilizan las máquinas de télex) que, en el mejor de los casos, sólo podían mover una herramienta fija de un punto a otro alrededor del objeto sobre el cual estaban trabajando.

El siguiente paso de su desarrollo fue conseguir que pudieran cambiar de herramienta en el transcurso de la faena. Esto se logró mediante la utilización de un "carrusel" o soporte de herramientas rotatorio; todas ellas tenían fijaciones idénticas, que se podían seleccionar y ajustar al soporte de herramientas bajo el control del programa.

Incluso con esta refinación, una máquina determinada sólo era capaz de realizar un único tipo de tarea: un torno seguía siendo un torno, aun cuando pudiera hacer todos los trabajos de tornería requeridos para un proceso determinado. Por esa misma época se estaban desarrollando manos y brazos accionados por control remoto para trabajar en medios peligrosos: bajo el océano, por ejemplo, o en laboratorios donde se manipulaban elementos radiactivos. Estos dispositivos manipuladores eran meras extensiones de las manos del operario, pero enseguida se comenzaron a utilizar ordenadores para controlarlos directamente. Los robots que se han desarrollado después son, aplicando un término más preciso, "brazos robot", pues consisten en un soporte de herramientas montado sobre un brazo extensible o articulado.

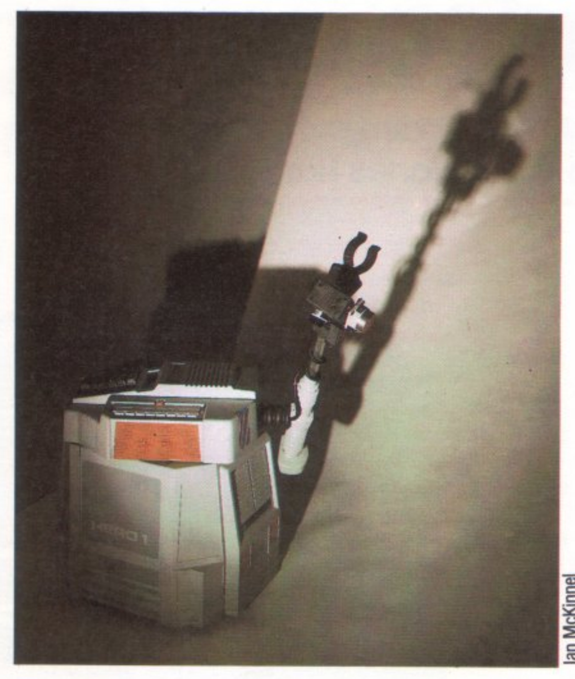
Si deseamos comprender cómo se programan los robots, primero debemos considerarlos en relación al espacio en el cual operan. La mayoría de los robots industriales ocupan una posición fija, de modo que el espacio será una esfera aplanada en su parte inferior, y podemos pensar en la cuestión del control del robot como un simple ejercicio de geometría tridimensional. El centro del esferoide será la articulación de los "hombros" del robot, y el radio será la longitud del brazo extendido, medido desde el "hombro" hasta la punta de los "dedos": la uña o soporte de la herramienta. Cualquier punto dentro de este espacio se puede expresar como tres coordenadas: por ejemplo, como distancias norte-sur, este-oeste y arriba-abajo, desde una posición cero o "punto de referencia". En este caso las coordenadas se denominan "cartesianas", en honor del filósofo y matemático francés René Descartes (1596-1650). Alternativamente, la posición se puede expresar por coordenadas esféricas. En lenguaje llano esto equivaldría a decir: "a una distancia de dos metros en dirección nordeste y treinta grados sobre la horizontal". En este caso el punto de referencia es el "hombro" del robot.

No obstante, el problema de programar al robot



Héroe del cine
R2D2, el cautivador robot de *La guerra de las galaxias*, en realidad estaba gobernado por un operador humano. Su diseño, sin embargo, reflejaba el aspecto que, según suele creerse, debe tener un robot

Un robot a pilas
El Hero-1 es un robot a pilas totalmente autocontenido que combina algunas de las funciones de una tortuga con la capacidad de manipulación de un brazo robot. Constituye un sistema por ordenador notablemente flexible, con configuraciones tan avanzadas como sintetizador de voz, sensores de nivel de luz, entrada auditiva y (debido a que es móvil) un enfocador ultrasónico de alto alcance que también actúa como detector de movimiento



Ian McKinnel

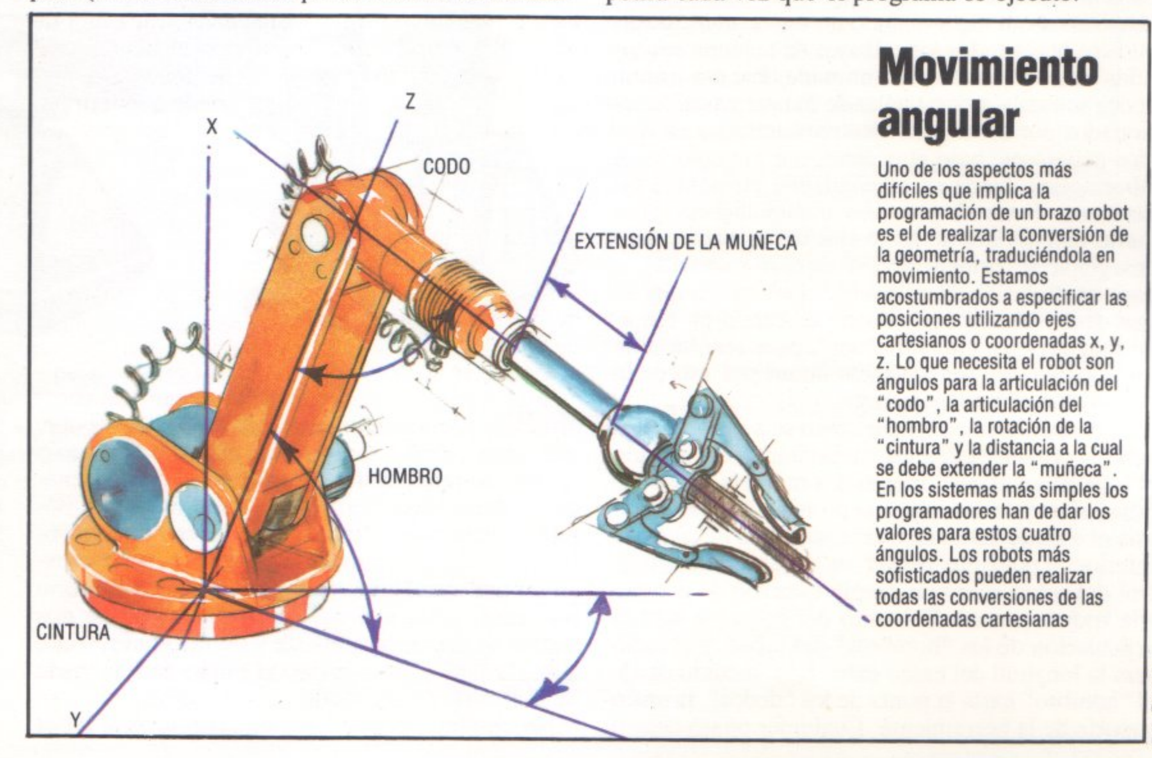
implica darle una serie de instrucciones acerca del movimiento desde un lugar hacia otro, de modo que existe aún un tercer procedimiento de determinar la posición del soporte de herramienta. Denominado *posicionamiento punto a punto*, este método requiere que el punto de referencia se mueva con el soporte de herramienta.

Por lo general, el margen de precisión de los robots industriales es de un milímetro. Incluso los modelos más sencillos (que se pueden utilizar con cualquier ordenador personal que posea una salida en paralelo de ocho bits) tienen un margen de precisión de dos milímetros.

Existen dos métodos generalmente aceptados para accionar los brazos robot. Para aquellos de poca carga útil, son suficientes los motores paso a paso (motores eléctricos que se mueven en un margen preestablecido cada vez que se les aplica corriente, como los que se emplean en las unidades de disco para colocar en posición la cabeza de lectura-escritura). Pero para los brazos robot que se utilizan en una cadena de producción, donde se necesita maniobrar pesos mucho mayores, es mucho más común emplear arietes hidráulicos para mover las diversas partes del brazo alrededor de su fulcro (los puntos alrededor de los cuales rotan). Resulta bastante sencillo medir el volumen de fluido hidráulico que pasa por los arietes y deducir del mismo el movimiento al otro extremo, de acuerdo con las exigencias operacionales de precisión.

Los robots industriales contienen un miniordenador construido especialmente (o, en los modelos más recientes, un microordenador de gran capacidad) y cuya exclusiva función es controlar el brazo y ejecutar un lenguaje de programación diseñado con esa finalidad. Dado que no se precisa otra exigencia que indicar coordenadas e impartir órdenes simples como CLOSE GRIPPER (cerrar uña) u OPEN GRIPPER (abrir uña), el lenguaje de programación no contiene instrucciones para manipular textos. A las instrucciones del programa se les da entrada a través de un teclado numérico agrandado acoplado al ordenador mediante un largo "cordón umbilical", de modo que el operador se pueda mover alrededor del brazo robot mientras da entrada a las instrucciones. Las versiones más avanzadas de estos *paneles colgantes*, como se denominan, incluyen una palanca de mando de precisión.

Otro procedimiento de programación, conocido como *Follow me* (Sígame), es especialmente útil para tareas que no precisen una colocación exacta de la herramienta, como cuando se pinta con una pistola pulverizadora. En este caso, el brazo del robot posee un dispositivo que permite al operador empuñar el soporte de herramienta, desplazarlo de manera exacta alrededor de la posición de trabajo y dar entrada a estos movimientos directamente en la memoria del ordenador. El robot, entonces, los repetirá cada vez que el programa se ejecute.



Movimiento angular

Uno de los aspectos más difíciles que implica la programación de un brazo robot es el de realizar la conversión de la geometría, traduciéndola en movimiento. Estamos acostumbrados a especificar las posiciones utilizando ejes cartesianos o coordenadas x, y, z. Lo que necesita el robot son ángulos para la articulación del "codo", la articulación del "hombro", la rotación de la "cintura" y la distancia a la cual se debe extender la "muñeca". En los sistemas más simples los programadores han de dar los valores para estos cuatro ángulos. Los robots más sofisticados pueden realizar todas las conversiones de las coordenadas cartesianas

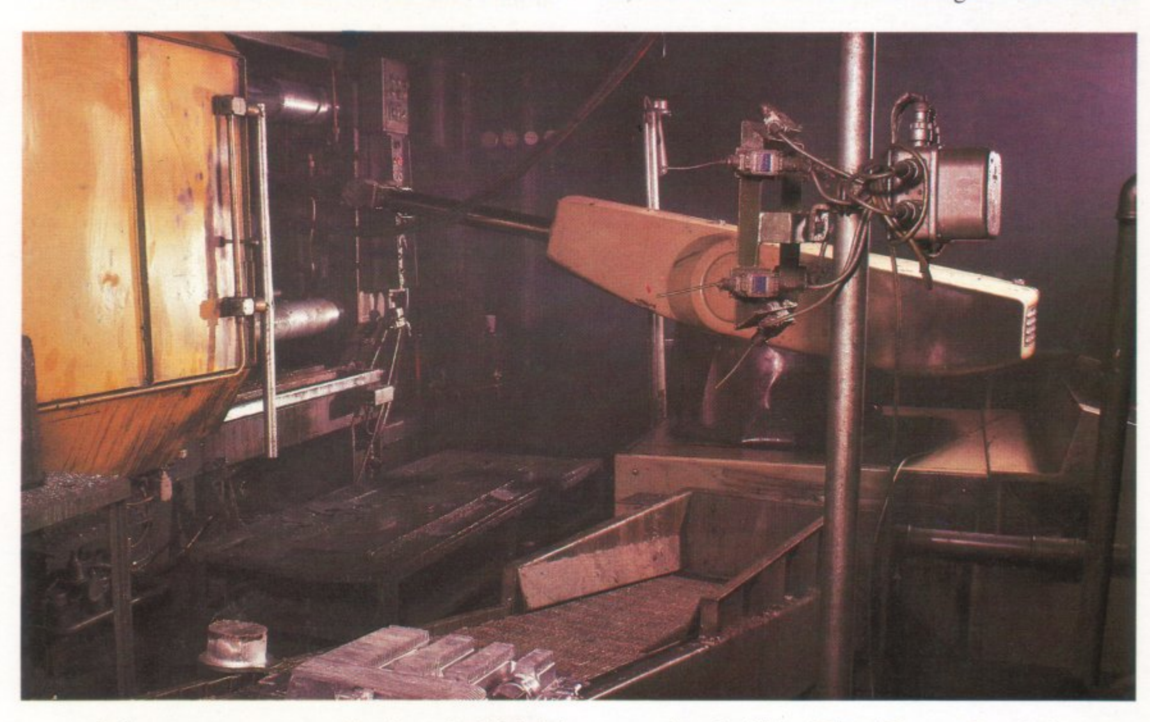
Bob Freeman

En todos estos métodos, la posición que se define es la del soporte de herramientas. El operador no necesita preocuparse de las posiciones relativas de las secciones individuales del robot: el lenguaje de programación incorporado en el ordenador de control del robot calcula cuáles deberían ser. También efectúa toda la optimización necesaria, asegurándose de que la herramienta se desplace de un lugar a otro por el camino más corto posible. La orientación del soporte de herramienta se controla de manera automática, manteniendo posiciones relativas tanto horizontales como verticales, a menos que se lo instruya en otro sentido. La velocidad del movimiento de punto a punto también es automática: el soporte de herramienta se desengancha despacio, se mueve rápidamente hasta acercarse al punto de destino y después vuelve a avanzar despacio para reengancharse a la pieza en el nuevo lugar.

Los robots de los que hemos hablado hasta ahora sólo son capaces de una "obediencia ciega", repitiendo la misma tarea exactamente en la misma localización, sin inmutarse por las influencias externas. Se los utiliza fundamentalmente en la industria de la ingeniería, en especial para la producción de vehículos motorizados. Ya hace mucho tiempo que ésta se ha organizado en cadenas de producción, en las cuales el componente o el vehículo parcialmente completo está siempre localizado con precisión en el espacio y el tiempo. Esto es de vital importancia para el buen funcionamiento de un proceso de fabricación por robot, porque si el componente no estuviera situado de forma correcta, el robot no adaptaría su movimiento en la debida forma. Con el objeto de superar este inconveniente, se pueden acoplar diversos sensores al soporte de herramienta. El más sencillo de estos sensores puede ser un microinterruptor convencional. En el programa de control se pueden incorporar planes de contingencia —por ejemplo, una orden WAIT (esperar)— para que se ejecuten en el caso de que el interruptor no haga contacto con la pieza; pero la aplicación de planes más sofisticados requeriría la intervención humana.

Una alternativa a la detección por presión implicaría la utilización de un sensor de luz. Si se colocara una fuente luminosa de modo que la pieza hiciera que ésta quedara oscurecida para el sensor del soporte de herramientas, éste se podría detener antes de que llegara al punto de colisión, se podría poner en modalidad WAIT hasta que la pieza estuviera bien colocada, y después permitir que continuara. Por supuesto, esto tampoco sería absolutamente seguro, y en las situaciones para las que se requiere una fiabilidad total se puede instalar un sistema de reconocimiento de imagen basado en cámaras de televisión CCD (*Charge-Coupled Device*: dispositivo de carga acoplada). Estas cámaras centran la imagen de manera precisa en un microchip de procesamiento por matriz (un chip dividido en cien o más fotosensores individuales, cada uno de los cuales puede registrar no sólo el blanco o el negro sino también una gama de tonos intermedios). Cada sensor individual puede requerir un byte de memoria para definir el contraste en la escala del gris. Inicialmente cada objeto se "fotografía" cierto número de veces y un programa de aprendizaje calcula el promedio de los valores. Al tiempo de ejecución, la cámara CCD toma del objeto una imagen, que luego se compara con la imagen de referencia de la memoria. Si las dos imágenes concuerdan, entonces la operación puede seguir adelante. Por este método se puede verificar que la pieza sea la correcta y su disposición la adecuada.

Este sistema de tratamiento de la imagen también se emplea para la selección de componentes en una "bolsa mezclada". Esta aplicación de "recogida y colocación", como se denomina, está siendo cada vez más utilizada para los robots pequeños como complemento de una cadena de producción regular. Además de en el proceso de producción propiamente dicho, los robots industriales se emplean en las etapas de prueba y control de calidad, a menudo a pares para posibilitar un mayor grado de flexibilidad en la colocación del producto en su posición correcta.



Jornada en la fábrica
Los brazos robot, como el que en la fotografía vemos trabajando en un taller de fundición, están tomando cada vez más bajo su cargo las tareas sucias, peligrosas y repetitivas de la industria. La limpieza de las piezas fundidas previa a su introducción en las máquinas constituye un buen ejemplo. La fundición, recién moldeada, es excesivamente caliente para las manos humanas y, por lo tanto, se colocaría a un lado hasta que se enfriara. Sin embargo, el robot no es sensible al calor, de modo que puede manipularla de inmediato y despacharla a la operación siguiente

Haciendo sonar el Vic

Una mirada a la generación de sonido del Vic-20...

El Vic-20 fue uno de los primeros ordenadores personales que salieron al mercado. Como consecuencia de ello, sus configuraciones pueden parecer poco satisfactorias en comparación con ordenadores más recientes. Además, Commodore no facilita de manera especial la creación de sonido ni de programas de música, ya que el BASIC del Vic-20, así como el BASIC del Commodore 64, no poseen órdenes relacionadas específicamente con el sonido. Todo el control de sonido se consigue mediante una serie de órdenes POKE en posiciones de memoria. Este principio se aplica también al Commodore 64, y las técnicas que aquí describimos para el Vic-20 le serán útiles al usuario del citado ordenador. El grado de control de sonido disponible se limita a volumen (equivalente a envoltura de A = D = R = 0), frecuencia en tres osciladores y un generador de ruidos. La salida sólo es posible a través del altavoz del televisor. Además, debido a las imprecisiones inherentes a la forma en que el Vic-20 selecciona las frecuencias, es imposible obtener la altura correcta para todas las notas de la escala musical.

Con sólo estas capacidades, el Vic-20 se muestra muy limitado si se quiere crear música; aunque, pensando mucho, con paciencia y con un poco de conocimiento de programación en BASIC, estas limitadas configuraciones se pueden utilizar para crear "melodías" de dos y tres acordes.

Control de sonido

El Vic-20 viene con tres osciladores de ondas cuadradas y un generador de ruidos. Cada oscilador abarca aproximadamente tres octavas de sonido, con la frecuencia compensada del siguiente modo:

Osc.1	Osc.2	Osc.3	Escala de frec. (Hz)	Octava
•			(65,41-123,47)	1
•	•		(130,81-246,94)	2
•	•	•	(261,63-493,88)	3
	•	•	(523,25-987,77)	4
		•	(1046,5-1975,53)	5

Esta distribución le permite al usuario cubrir cinco octavas en total con al menos un oscilador disponible en cada octava. La octava 3, que empieza en *do mayor* y contiene la referencia estándar *la* en 440 Hz, está disponible en los tres osciladores.

El control de los osciladores se ejerce modificando los contenidos de cinco posiciones de memoria de la siguiente manera:

Posición de memoria	Oscilador
POKE 36874,X	1
POKE 36875,X	2
POKE 36876,X	3
POKE 36877,X	ruido

En cada caso X es un número entero entre 135 y 241 (0 apaga ese oscilador), que se refiere a una tabla de valores de notas equivalentes reseñada en el folleto que acompaña al Vic-20. Para que se pueda escuchar la frecuencia seleccionada se debe establecer el nivel de volumen del siguiente modo:

```
POKE 36878,V
```

donde V se puede establecer entre 0 (apagado) y 15 (alto) afectando a los tres osciladores y al ruido. Por ejemplo:

```
POKE 36874,219:POKE 36875,219:POKE
  36876,219:POKE 36878,7
```

Esto hace tocar la referencia *la* a 440 Hz en el oscilador 1, *la* una octava más alta en el oscilador 2 y *la* una octava aún más arriba en el oscilador 3, todas ellas a un volumen medio de 7. ¡No se olvide de colocar POKE cada posición en 0 para apagarlas!

Notas y pausas

Sin una duración para cada nota y sin las pausas adecuadas entre ellas, en una secuencia de notas éstas se confundirían entre sí. Para facilitar estos períodos de "espera", se puede emplear uno de dos métodos para lograr que el ordenador "marque el compás" entre un POKE y otro. El primer procedimiento es el de los bucles FOR...NEXT, donde la pausa se sincroniza mediante un bucle vacío largo como:

```
10 POKE 36878,7
20 POKE 36876,203
30 FOR P = 1 TO 200
40 NEXT P
50 POKE 36878,0
60 POKE 36876,0
```

Esta secuencia de órdenes hace que se toque la nota *re #* para 200 operaciones FOR...NEXT. Sin embargo, la exactitud de este método depende de una cuidadosa sincronización externa del bucle. Una forma más sencilla y más elegante de establecer las duraciones y las pausas consiste en utilizar el reloj incorporado del Vic-20, que cuenta en sesentavos de segundo (*jiffys*) y al que se puede hacer referencia dentro de un programa empleando la variable TI. Ésta es extremadamente útil, ya que una orden se puede construir como para que "espere" durante un lapso medido con suma precisión, de este modo:

```
10 POKE 36878,7
20 POKE 36876,203:RE = TI
30 IF TI-RE < 15 THEN 30
40 POKE 36878,0
50 POKE 36876,0
```

Estas órdenes tocan la misma nota que antes pero durante un período de 15 *jiffys* (la cuarta parte de un segundo). Cuando se enciende el sonido, RE se establece en el valor de TI. La línea 30 cuenta 15 *jiffys* antes de proseguir a la línea 40. Se pueden construir melodías utilizando el mismo principio para hacer una pausa antes de tocar una nota diferente, y así sucesivamente. La próxima vez que nos ocupemos del Vic-20 en nuestra serie "Sonido y luz", estudiaremos cómo tocar melodías.



Iluminando el Dragon

... y a la capacidad para gráficos del Dragon 32

El ordenador Dragon 32 incorpora un dialecto particular de BASIC que se conoce como *Microsoft Extended Colour Basic.* Además del Dragon, existen en el mercado otros varios ordenadores que poseen también esta versión de BASIC, entre los que destaca la gama de ordenadores en color Tandy. El BASIC Microsoft es sencillo de utilizar y dispone de una buena gama de órdenes para trazar líneas, círculos y otras formas geométricas. Una vez dibujadas, estas formas se pueden colorear, obteniendo con un escaso esfuerzo de programación unas visualizaciones en pantalla notables.

El Dragon 32 posee siete niveles de resolución, proporcionándole al usuario la capacidad de trabajar con la pantalla dividida en 512 puntos individuales en el nivel más bajo, y en hasta 49 152 puntos en el nivel más alto. Hay ocho colores disponibles, pero al trabajar con alta resolución la elección se puede limitar a cuatro o incluso a dos colores.

Modalidades de resolución

La pantalla normal de 16 filas por 32 columnas de caracteres conforma el nivel de resolución más bajo, y la orden PRINT@ permite colocar al carácter en cualquiera de las 512 posiciones de pantalla. Además del juego de caracteres normal, existen 16 caracteres para gráficos de baja resolución disponibles en ocho colores.

La siguiente modalidad de resolución divide a la pantalla en 32 filas y 64 columnas. En esta modalidad, el tamaño de cada cuadrado es, por lo tanto, equivalente a la cuarta parte del de un carácter normal. Los puntos de este tamaño se pueden trazar en la pantalla mediante la orden SET y se pueden borrar mediante la orden RESET.

Las dos modalidades anteriores se pueden visualizar al mismo tiempo y se llaman pantallas de texto de baja resolución. Existen asimismo cinco niveles de pantallas de alta resolución, pero éstos no se pueden visualizar simultáneamente o en las pantallas de bajo nivel. Las cinco modalidades de alta resolución ofrecen opciones basadas en el estándar de resolución y el número de colores disponibles, y se seleccionan utilizando la orden PMODE.

PMODE	Resolución	Colores disponibles
0	128*96	2
1	128*96	4
2	128*192	2
3	128*192	4
4	256*192	2

Existe, por supuesto, una interrelación entre resolución, color y la cantidad de memoria necesaria para almacenar la información en pantalla, y esto se debe tener en cuenta al escribir programas largos en BASIC que también utilicen visualizaciones de alta resolución.

Aunque sólo disponga de un número limitado de colores en alta resolución, el Dragon posee la capacidad de seleccionar uno de dos juegos de colores. Esto se consigue mediante la orden SCREEN. Por ejemplo, SCREEN 1,0 selecciona una pantalla de alta resolución y el juego de colores 0; SCREEN 1,1 selecciona también una pantalla de alta resolución pero con un juego de colores alternativo.

PAINT
Esta orden resulta muy útil para ayudar al programador a crear imágenes interesantes. La utilización de PAINT hace que el ordenador comience a colorear la pantalla desde un punto determinado hasta llegar a una línea limítrofe. Esto significa que se puede rellenar fácilmente círculos, triángulos y cualquier forma cerrada.

DRAW
DRAW imita el movimiento del lápiz sobre la pantalla, permitiéndole al usuario trazar líneas en una de cuatro direcciones. Con la orden DRAW también se puede hacer girar o agrandar la imagen terminada.

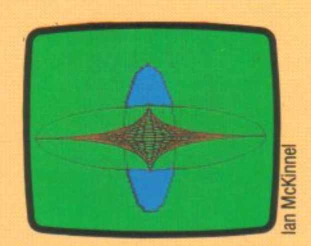
GET y **PUT**
GET instruye al ordenador para que almacene en su memoria una visualización en pantalla, y PUT hace que dicha visualización se vuelva a imprimir en la pantalla.

PSET y **PRESET**
Estas órdenes son los equivalentes para alta resolución de SET y RESET que ya hemos analizado antes y que encienden o apagan un punto determinado de la pantalla. También se puede determinar el color del punto.

LINE
La orden LINE hace que, en alta resolución, dos puntos especificados se unan entre sí por una línea recta.

CIRCLE
CIRCLE permite dibujar círculos de alta resolución con un centro y un radio determinados. También cabe dibujar porciones de un círculo completo para formar arcos y se puede condensar la forma circular para producir elipses.

El Dragon 32 es un ordenador que posee muchas órdenes avanzadas para ayudar a la programación de gráficos. Es más adecuado para aplicaciones que impliquen visualizaciones estáticas que para aquellas que requieran una acción de movimiento rápido. Especialmente las órdenes de la modalidad de alta resolución hacen del Dragon 32 un ordenador ideal para los niños de mentalidad emprendedora. El principal inconveniente es su incapacidad para visualizar simultáneamente en la pantalla texto y gráficos de alta resolución. Esto significa que no se puede utilizar para visualizar datos estadísticos en forma de diagramas de barras o cuadros.



Ian McKinnel

Orden de color
Esta visualización constituye un ejemplo típico de los efectos que se pueden obtener en un Dragon utilizando algunas de sus órdenes de alto nivel

Alta resolución
He aquí un breve programa para el Dragon 32 que demuestra algunas de sus capacidades de alta resolución. El programa emplea PMODE 3; ésta no es la modalidad más alta, pero admite cierta utilización de color

```
10 PCLS:PMODE3,1
20 SCREEN 1,0
30 COLOR 0,1
40 FOR X = 0 TO 127 STEP 10
50 LINE(X,85)-(127,85-X/3),
   PSET
60 LINE(X,85)-(127,85+X/3),
   PSET
70 LINE (255-X,85)-(127,85-
   X/3), PSET
80 LINE (255-X,85)-(127,85+
   X/3),PSET
90 NEXT X
100 CIRCLE(127,85),128,4,0.3
110 CIRCLE(127,85),30,4,3
120 PAINT(130,30),3,4
130 PAINT(130,130),3,4
140 GOTO 140
150 END
```

Ordenando la baraja

En la mayoría de los programas resulta esencial su capacidad para clasificar la información, y existen muchas formas de hacerlo

Clasificación burbuja
Este diagrama ilustra la "clasificación burbuja" para una baraja reducida de 9 naipes (D corresponde a la carta Diez). La parte ordenada de la baraja va creciendo a cada pasada desde el extremo derecho. El 1 y el 2 debajo de la baraja indican los dos naipes que se están comparando en cada momento

```
Empezar clasificación
2 8 9 3 D 5 K 6 7  Empezar pasada 1
1 2
8 2 9 3 D 5 K 6 7
  1 2
8 9 2 3 D 5 K 6 7
    1 2
8 9 3 2 D 5 K 6 7
      1 2
8 9 3 D 2 5 K 6 7
        1 2
8 9 3 D 5 2 K 6 7
          1 2
8 9 3 D 5 K 2 6 7
            1 2
8 9 3 D 5 K 6 2 7
              1 2
8 9 3 D 5 K 6 7 2  Fin pasada 1
9 8 D 5 K 6 7 3 2  Fin pasada 2
9 D 8 K 6 7 5 3 2  Fin pasada 3
D 9 K 8 7 6 5 3 2  Fin pasada 4
D K 9 8 7 6 5 3 2  Fin pasada 5
K D 9 8 7 6 5 3 2  Fin pasada 6
Fin clasificación
```

Clasificación por inserción
Con la "clasificación por inserción", la parte ordenada de la lista va creciendo desde el extremo izquierdo. Los naipes se desplazan directamente hasta su posición correcta en la lista a medida que son revisados

```
Empezar clasificación
2 8 9 3 D 5 K 6 7
2 1
8 2 9 3 D 5 K 6 7
2   1
9 8 2 3 D 5 K 6 7
    2 1
9 8 3 2 D 5 K 6 7
2       1
D 9 8 3 2 5 K 6 7
      2   1
D 9 8 5 3 2 K 6 7
2           1
K D 9 8 5 3 2 6 7
        2     1
K D 9 8 6 5 3 2 7
        2       1
K D 9 8 7 6 5 3 2
Fin clasificación
```

La clasificación es una de las operaciones por ordenador que se utilizan más ampliamente, pero es una tarea para la cual estas máquinas, por sus propias normas de funcionamiento, son sumamente ineficaces. De acuerdo con la investigación operativa, se invierte en la clasificación entre el 30 y el 40 % del tiempo informático, y si sumáramos las otras tareas que van unidas a ella —intercalación de datos y búsqueda de ítems específicos—, es posible que la cifra se elevara a más del 50 %.

Es probable que los programadores destinen tanto tiempo a inventar algoritmos de clasificación (métodos generales para resolver problemas) como el que invierten los ordenadores en efectuar la clasificación real. Los métodos de clasificación avanzados son muy difíciles de analizar, pero nos resultará bastante fácil comprender los procedimientos más sencillos que utilizan los ordenadores para clasificar los datos, tomando como ejemplo la clasificación de una baraja de naipes.

Coloque sobre una mesa 13 naipes del mismo palo. Póngalos en línea, sin seguir ningún orden determinado, pero sin que ni el As ni el Dos queden en el extremo derecho de la línea. Los naipes se han de clasificar por orden descendente (Rey, Dama, Valet...As), comenzando por la izquierda. Para nosotros ésta es una tarea casi trivial y requiere tan poco esfuerzo mental que es difícil describir exactamente cómo lo haríamos. No obstante, si se especificara que sólo se puede mover una carta cada vez, que no se puede colocar un naipe encima de otro y que las cartas deben cubrir la menor superficie posible de la mesa, la tarea ya sería mucho menos trivial y resultaría difícil determinar un método eficaz. En esta analogía los naipes son datos, la superficie máxima cubierta corresponde a la memoria del ordenador requerida y usted es el programa. ¿Cómo resolvería el problema?

1) Coloque una moneda debajo del naipe situado más a la izquierda, para que actúe como indicador de posición y para que le recuerde en qué punto de la clasificación se encuentra. Compare el naipe marcado con el naipe situado a su derecha. ¿Están en orden descendente? Si no lo están, invierta sus posiciones, dejando la moneda en el mismo sitio, y obedeciendo la regla de mover sólo una carta cada vez y de no colocar un naipe encima de otro. Observe lo que debe hacer para invertirlos.

2) Cuando los dos naipes estén en orden, desplace la moneda un lugar hacia la derecha y repita el paso 1. Ahora se encuentra en un bucle que terminará cuando coloque la moneda en el lugar situado más a la derecha. Alcanzar esta posición se conoce como efectuar una "pasada" a través de las cartas.

3) Finalizada la primera pasada, eche una mirada a los naipes. El As, que es la carta más baja de la baraja, se ha abierto camino hasta el extremo derecho de la línea y, por lo tanto, se halla en el sitio correcto. Si hace otra pasada a través de los naipes, siguiendo el procedimiento detallado en los pasos 1 y 2, el Dos se desplazará hasta el lugar que le corresponde. Esto se repetirá, de una pasada a otra, hasta que toda la baraja se encuentre en orden descendente.

Es posible que haya observado diversos inconvenientes en este método. Es muy tedioso; no económico, ya que la sola acción de intercambiar las posiciones de dos naipes requiere efectuar tres operaciones diferentes; y, sobre todo, muchas de las comparaciones realizadas entre cartas distintas son innecesarias. Por ejemplo, al cabo de una pasada el As está en el lugar que le corresponde, de modo que no tiene ningún sentido desplazar la moneda a la posición 13 (donde, en todo caso, no existe comparación posible). En la segunda pasada, dado que el naipe situado a la derecha está en el lugar correcto, no hay necesidad de desplazar el señalador hasta la posición 12. En general, cada pasada terminará un lugar a la izquierda respecto al punto donde finalizó la pasada anterior.

Saber dónde se debe detener es otro problema. Un ordenador seguiría comparando naipes indefinidamente a menos que le dijéramos que parara. La única regla segura es detenerse después de una pasada sin inversiones. En otras palabras, si usted ha pasado a través de los datos sin tener que modificar su orden, se deduce que están ordenados.

El método de clasificación que hemos expuesto se denomina *clasificación burbuja*. Entre sus ventajas figuran: empleo de técnicas de programación simples, poca utilización de memoria extra y razonable eficacia con pequeñas cantidades de datos ordenados parcialmente. Éstos son los criterios en virtud de los cuales se debe juzgar un algoritmo de clasificación, si bien cuando los datos a clasificar son muchos, se ha de sacrificar la velocidad para economizar memoria, simplemente porque puede que la memoria del ordenador no tenga capacidad tanto para los datos en bruto como para una copia clasificada. Por este motivo, ignoraremos los algoritmos que requieran tomar datos de una matriz y desplazarlos hasta la posición clasificada de una segunda matriz. El segundo método de clasificación simple se base más directamente en la manera en que nosotros clasificaríamos las cartas.

1) Vuelva a colocar los naipes mezclados y coloque una moneda de una peseta debajo del segundo naipe empezando por la izquierda. Cualquiera que sea la carta bajo la cual esté colocada la moneda a cada pasada, la llamaremos "naipe de la peseta".

2) Empuje el naipe de la peseta fuera de la línea, dejando un lugar vacío, y coloque una moneda de cinco pesetas debajo del naipe situado inmediatamente a la izquierda. A este naipe lo llamaremos "naipe de las cinco pesetas".



Gran slam

La "clasificación burbuja" se puede ilustrar mediante una baraja de naipes que han de clasificarse de modo que el Rey (K) acabe estando en el extremo izquierdo y el As en el derecho. Primero se comparan los dos naipes situados más a la izquierda y, como se descubre que no están en orden, se invierten. Luego se comparan el segundo y el tercer naipe y, nuevamente, se invierten. En la quinta comparación, este método de clasificación ha recogido al As y en todas las comparaciones siguientes éste se va desplazando por inversión, de izquierda a derecha, hasta que al final de la primera "pasada" se ha abierto su camino, "burbujeando", hasta el extremo derecho. Sin embargo, podrían necesitarse 12 pasadas antes de que los naipes estuvieran en orden

Tony Lodge

3) Compare el naipe de la peseta con el naipe de las cinco pesetas. Si están en orden, vuelva a empujar a su lugar el naipe de la peseta y comience el paso 4. Si no están en orden, empuje entonces el naipe de las cinco pesetas hasta el lugar vacío y desplace la moneda de cinco pesetas un lugar hacia la izquierda para señalar un naipe de cinco pesetas nuevo. (Si la carta de las cinco pesetas está situada en el extremo izquierdo, esto no se aplica, de modo que coloque el naipe de la peseta en el lugar vacío y continúe por el paso 4.)

Compare este naipe de cinco pesetas con el naipe de la peseta (el desplazado). Ahora repita el paso 3 hasta hallar la posición correcta para el naipe de la peseta.

4) Desplace la peseta un puesto hacia la derecha y repita los pasos 2 y 3. Cuando ya no pueda desplazar la peseta hacia la derecha, los naipes estarán todos en orden.

Este método se denomina *clasificación por inserción* y se asemeja mucho a la forma en que solemos ordenar una baraja de naipes. Aunque es un poco más difícil de programar que una clasificación burbuja, es un método mucho más eficaz. Más adelante analizaremos algunos algoritmos más complejos para clasificación de datos.

```
9  REM ********************************
10 REM * ALGORITMOS DE CLASIFICACION *
11 REM ********************************
100 INPUT "CUANTOS ITEMS A CLASIFICAR";LT
150 IF LT <3 THEN LET LT = 3
200 LET LT = INT(LT)
250 DIM R(LT) ,C(LT)
300 LET Z = 0:LET Q = 0:LET P = 0
350 LET I = 1:LET O = 0: LET II = 2:LET TH = 2
400 INPUT "CUANTAS CONDICIONES";N
450 FOR CT = I TO N
500 GOSUB 4000
550 FOR SR = I TO TH
600 GOSUB 5000
650 PRINT:PRINT:PRINT:PRINT
700 PRINT "CONDICION #";CT+SR/10
750 INPUT "PULSE RETURN PARA COMENZAR
    CLASIFICACION":A$
800 PRINT "LA LISTA NO CLASIFICADA ES"
850 GOSUB 3000
900 ON SR GOSUB 6000,7000
950 PRINT "LA LISTA CLASIFICADA ES"
1000 GOSUB 3000
1050 NEXT SR
1100 NEXT CT
1150 END
2999 REM ********************************
3000 REM * IMPRIMIR LA LISTA *
3001 REM ********************************
3100 FOR K = I TO LT
3200 PRINT R(K);
3300 NEXT K
3400 PRINT
3500 RETURN
3999 REM ********************************
4000 REM * GENERADOR ALEATORIO *
4001 REM ********************************
4100 RANDOMIZE
4200 FOR K = I TO LT
4300 LET C(K) = INT(100*RND)
4400 NEXT K
4500 RETURN
4999 REM ******************************
5000 REM * GENERADOR ALEATORIO *
5001 REM ******************************
5100 FOR K = I TO LT
5200 LET R(K) = C(K)
5300 NEXT K
5400 PRINT:PRINT
5500 RETURN
5999 REM ***********************
6000 REM * BURBUJA *
6001 REM ***********************
6050 PRINT "CLASIFICACION BURBUJA - ADELANTE !!!!! "
6100 FOR P = LT - I TO I STEP - I
6150 LET F = -I
6200 FOR Q = I TO P
6250 LET Z = Q+I
6300 IF R(Q)<R(Z) THEN LET D = R(Q) :
     LET R(Q) = R(Z): LET R(Z) = D: LET F = O
6350 NEXT Q
6400 IF F = -I THEN LET P = I
6450 NEXT P
6500 PRINT "CLASIFICACION BURBUJA - STOP !!!!! "
6550 RETURN
6999 REM **********************
7000 REM * INSERCION *
7001 REM **********************
7050 PRINT "CLASIFICACION POR INSERCION
     - ADELANTE !!!!! "
7100 FOR P = II TO LT
7200 LET D = R(P)
7300 FOR Q = P TO II STEP - I
7400 LET R(Q) = R(Q - I)
7500 IF D< = R(Q) THEN LET R(Q) = D:LET Q = II
7600 NEXT Q
7700 IF D> R(I) THEN LET R(I) = D
7800 NEXT P
7850 PRINT "CLASIFICACION POR INSERCION - STOP !!!!! "
7900 RETURN
```

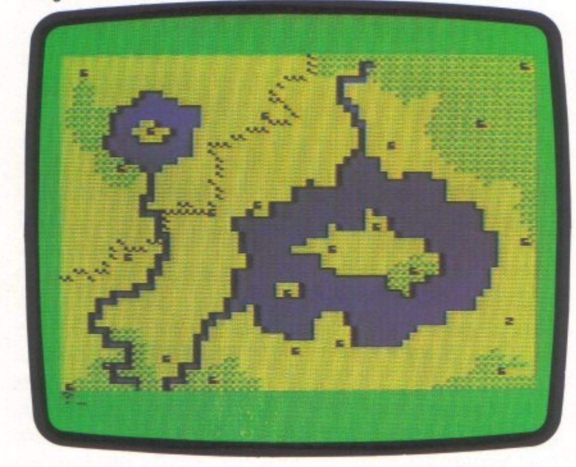
Clasificación a gran velocidad
Este programa en BASIC muestra la diferencia, en cuanto a eficacia, entre una "clasificación burbuja" y una "clasificación por inserción". El código se ha escrito teniendo siempre presente la velocidad, de modo que no hemos documentado la operación de las rutinas. El listado se debería poder ejecutar en la mayoría de las máquinas, pero lea en p. 215 los "complementos" referentes a ON…GOSUB y en p. 175 los relativos a RND y RANDOMIZE

Juegos de laberinto

A la gente siempre le han atraído los laberintos, y los juegos de este tipo creados por ordenador conservan toda su fascinación

Los laberintos siempre han sido una fuente de diversión y atracción tanto para los jóvenes como para los mayores, trátese de laberintos tan grandes como para perderse en ellos o tan pequeños que quepan en la palma de la mano. De hecho, el laberinto se ha convertido en la base de una gran variedad de juegos por ordenador, que abarcan desde una panorámica aérea muy sencilla y en dos dimensiones hasta laberintos tridimensionales sumamente complejos. Los de esta última clase en realidad simulan el aspecto de un laberinto visto desde su interior, de modo que el jugador se imagina que se halla dentro de uno verdadero. Para ayudar al jugador a que se haga una composición de lugar, o incluso para confundirlo aún más, algunas de estas imágenes de laberintos tridimensionales se combinan con vislumbres de una panorámica aérea de él.

Ring of darkness



Ring of darkness (Círculo de oscuridad)
A pesar de que este juego para el Dragon está catalogado como "de aventuras", contiene un laberinto tridimensional como uno de sus elementos principales. Fosos y escaleras le permiten al jugador moverse hacia arriba y hacia abajo

Siren city (Ciudad de sirenas)
Este juego para el Commodore 64 es un desarrollo del juego de "panorámica aérea" tradicional. Un coche de policía patrulla por una ciudad, con sus calles y sus edificios

Way out (Salida)
Con "Way out" se puede obtener en el Spectrum una realista imagen en tres dimensiones. Basta accionar mínimamente la palanca de mando para que la panorámica cambie

Así como los laberintos se han ido haciendo cada vez más sofisticados en cuanto a sus efectos visuales y sonoros, igualmente se les ha ido permitiendo a los programadores dejar volar su imaginación. Un jugador que quiera dar un relajante paseo a través de un laberinto habrá de evitar aquellos que esconden monstruos devoradores de hombres. Un ejemplo de esta clase de juegos es el *3D Glooper* (disponible para el Commodore 64), en el cual el jugador busca la salida del laberinto siguiendo unas baldosas especiales y puede ser atacado en cualquier momento por unos monstruos que ocupan toda la pantalla. La inminente aparición de una de estas criaturas se anuncia mediante un ruido cada vez más cercano de mandíbulas batientes.

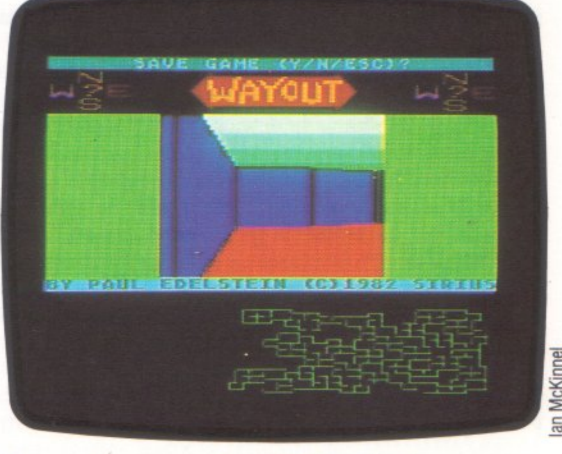
Atic Atac (Spectrum) es un juego de persecución totalmente animado en el cual el jugador puede asumir el papel de cualquiera de tres personajes distintos. El laberinto es una serie de fosos, escalinatas y grandes mazmorras a varios niveles que se deben recorrer contra reloj. Las mazmorras están habitadas por diversas criaturas y objetos ilustrados gráficamente.

Un programa que simula con notable fidelidad lo que uno siente realmente al viajar a través de un laberinto es *Way out* (Salida). Las imágenes están en una perspectiva tridimensional, y cuando el jugador mueve la palanca de mando unas fracciones hacia la izquierda o hacia la derecha, la escena se desplaza proporcionalmente en esa dirección.

Consideremos ahora algunas de las técnicas de programación básicas para construir laberintos.

Siren city

Way out



Ian McKinnel

Construyendo laberintos

La forma corriente de almacenar la información relativa a un laberinto consiste en utilizar una matriz bidimensional: L$(FILA,COLUMNA), por ejemplo. Cada celda de la matriz definiría las características de esa celda del laberinto. Se podría utilizar, por ejemplo, una serie de cuatro caracteres para representar sur, oeste, norte y este. "Cero" indicaría la ausencia de pared y "uno" la presencia de una muralla. En consecuencia, si L$(5,6) contuviera la variable "1011", indicaría que la celda de la fila 5, columna 6, está limitada por paredes por el sur, el

norte y el este. Para ahorrar espacio de memoria, la matriz podría ser numérica en vez de alfanumérica, y el número de cuatro dígitos podría considerarse como un número binario. En nuestro ejemplo anterior, las celdas que incluyen norte, este y sur contendrían el número 11 (1011).

Todas las celdas comenzarían con cuatro paredes. Mediante la generación al azar de la entrada desde cualquier lugar del perímetro, se podría escoger aleatoriamente, de entre cualquiera de las tres celdas adyacentes, la siguiente celda. Una vez escogida ésta, la secuencia continúa, seleccionando al azar una celda de entre cualquiera de las tres adyacentes, haciendo caso omiso de la celda de la cual proviniera usted.

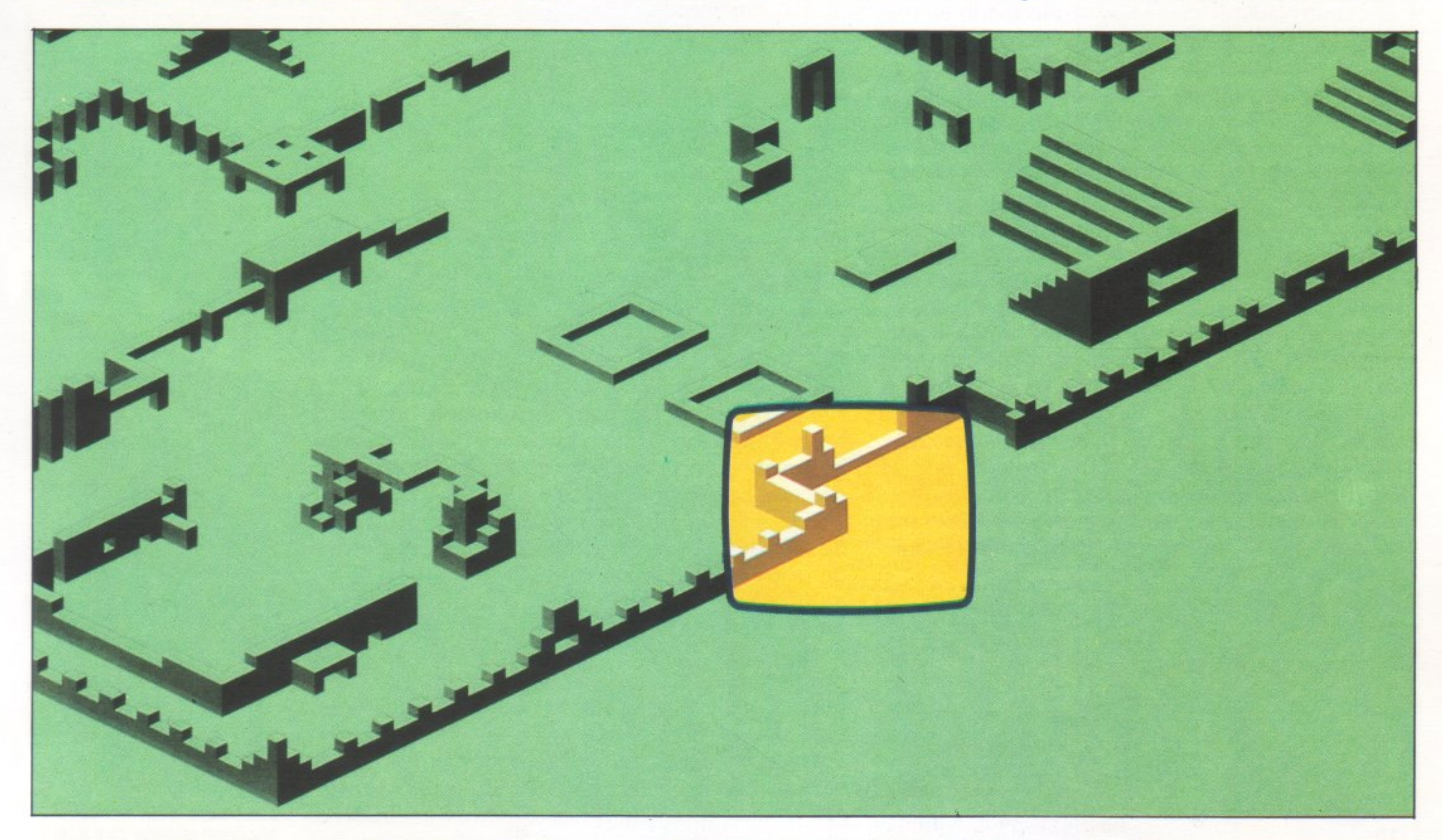
A medida que se escoge una nueva dirección, se quita la "pared" adecuada de la celda que se está por abandonar y de aquella a la que se va a entrar. Se deben efectuar verificaciones para asegurarse de que no se sale de los límites del laberinto (a menos que una celda determinada del perímetro sea el punto de salida) ni se crean circuitos cerrados (se debe poder acceder a todas las partes del laberinto desde cualquier punto).

Cuando uno se encuentra con una celda que posee menos de cuatro paredes (es decir, una celda que ya ha sido visitada), el programa debe escoger otra de las celdas adyacentes restantes. Si se han visitado todas las celdas adyacentes, el programa debe "retroceder un paso" hasta la celda visitada previamente y tomar una nueva bifurcación.

Existen 16 formas posibles de construir una celda: sin paredes, con paredes en todos los lados, con una sola pared (cuatro posibilidades), con paredes a los lados opuestos (dos posibilidades), con dos paredes adyacentes (cuatro posibilidades), y con tres paredes adyacentes (cuatro posibilidades).

Utilizando el número binario adecuado (0-15) para cada una de estas formas, se puede "hacer rotar" el número a izquierda o derecha para obtener la panorámica adecuada que observa el jugador. Por ejemplo, una pared norte se representaría por 2 (0010), una pared sur por 8 (1000), una pared este por 1 (0001) y una pared oeste por 4 (0100). Si el jugador que mirara al norte desde una celda con "una sola pared oeste" (4) girara para mirar al oeste, su panorámica estaría ahora limitada por una pared norte (porque mirar hacia adelante en una visualización tridimensional siempre es hacia el "norte"). Si el jugador se volviera hacia su izquierda (el oeste), mover el patrón de bits un lugar hacia la derecha proporcionaría la descripción que deseamos, o sea el binario pared oeste 0100 (decimal 4) se convierte en el binario 0010 (decimal 2: ¡una pared norte!). Los bits se mueven en la dirección opuesta al girar hacia la derecha, dos veces en una media vuelta. Es necesario, por supuesto, incluir un sistema para "recuperar" los bits que durante este proceso se pierden del extremo izquierdo o del derecho del medio byte, puesto que, de lo contrario, cada vez que el jugador girara dentro de una celda las características de identificación de ésta se modificarían. Una celda definida originalmente como 0011, por ejemplo (con paredes al norte y al este), ha de convertirse en 0110 si el jugador gira hacia la derecha, y en 1100 si da media vuelta.

En código de lenguaje máquina existen instrucciones especiales para hacer rotar los números binarios a derecha e izquierda. En BASIC, un número binario de cuatro bits expresado como decimal en la escala 0-15, se puede hacer girar hacia la izquierda multiplicando el número por dos y restando luego 15 si el resultado fuera mayor que 15. Para rotar hacia la derecha: dividir por dos si se trata de un número par, sumar 15 y dividir luego por dos en el caso de que fuera un número impar.



Ant attack (Ataque de las hormigas)
Cuando se ejecuta este juego en el Spectrum, la pantalla del ordenador actúa como una ventana abierta a un gran campo de juego parecido a un laberinto. A medida que se desarrolla el juego, la pantalla se desplaza, revelando detalles del escenario

Aquarius

Aunque fabricado por una empresa famosa por sus juguetes, el Aquarius es un ordenador eficaz y en toda regla

Con un procesador Z80 y su teclado estilo botón, el Mattel Aquarius es un microordenador en la línea del Spectrum. Sin embargo, en muchos aspectos es una máquina mucho más flexible, en gran medida porque sus diseñadores han sabido aprovechar muy bien su bus de ampliación incorporado.

A través de este bus se pueden conectar numerosos módulos de ampliación, desde pequeños paquetes de RAM de 4 Kbytes hasta un chasis de ampliación grande. Quizá el más útil de todos estos módulos sea el "chasis de ampliación pequeño", que posee dos ranuras para memoria extra o paquetes de programas, así como dos canales extra de sonido y dos controladores manuales. Conectando en una ranura un paquete de RAM de 16 Kbytes y, en la otra, un paquete de ROM patentado, como el Finplan, se puede obtener un sistema muy versátil.

La RAM de 4 Kbytes incorporada en la máquina no es muy generosa, pero con una ampliación de hasta 64 Kbytes, susceptible de lograr con el chasis de ampliación grande, se consigue una máquina tan potente como cualquier ordenador personal.

No obstante, el teclado y la visualización del Aquarius carecen de la calidad de las máquinas más grandes. El primero no posee barra espaciadora y la respuesta de las teclas no es ni muy rápida ni muy sensible, por lo cual no es apto para la escritura al tacto. La pantalla de 24 líneas de 40 caracteres, si bien es más grande que la de otros ordenadores, no es adecuada para su uso en pequeñas empresas.

La visualización dispone de 16 colores que se pueden utilizar tanto para el texto como para el fondo. Aunque carece de caracteres definibles por el usuario, posee 256 símbolos visualizables, que incluyen letras mayúsculas y minúsculas y una selección de símbolos gráficos. También se puede emplear como una pantalla de alta resolución de 320 × 192 pixels. La salida de la visualización es al televisor y no dispone de salida para monitor. La calidad es uniforme, con una notable tendencia hacia tonalidades azuladas y caracteres ligeramente borrosos, pero la imagen es continua y brillante, con una buena gama de color.

Esta máquina dispone de sonido, aunque carece de los sofisticados controles de envoltura y forma de onda que poseen otras. Lleva incorporado un BASIC Microsoft estándar, pero está previsto incluir un BASIC ampliado y un LOGO de Aquarius.

Uno de los accesorios más interesantes proyectados para el Aquarius es el sistema BSR X-10, que puede controlar una gama de aparatos domésticos. Este sistema permite controlar hasta 255 dispositivos eléctricos distintos en respuesta a las señales generadas por una unidad central. No se requiere ningún tendido de cables adicional, porque estas señales son en forma de impulsos enviados a través de la red eléctrica de la casa. Los impulsos no son lo suficientemente potentes como para que produzcan diferencias en la red de la corriente, pero un detector X-10 enchufado en cualquier toma de corriente puede captar el código y alterar la corriente suministrada a su aparato de acuerdo a la orden enviada.

La unidad controlada la programa el Aquarius por ciclos semanales, y durante esta operación no se puede usar el ordenador para otros fines. Siempre y cuando el programa preestablecido sea satisfactorio, el ordenador queda libre para emplearlo normalmente en cualquier otro momento.

La impresora Aquarius
Esta impresora utiliza un mecanismo de impresión térmico y, en consecuencia, requiere un papel térmico especial. Puede imprimir a una velocidad de 80 caracteres por segundo, a través de una anchura total de 40 columnas



Miniampliador
Este dispositivo incorpora dos conexiones para cartucho, permitiendo conectar simultáneamente un cartucho de programas y un paquete de memoria. También incorpora los dos "controladores manuales" y tres canales de sonido adicionales

Chris Stevens

El teclado del Aquarius
El teclado es uno de los puntos más débiles del Aquarius. A pesar de que se lo promociona como un teclado QWERTY "estándar", apenas si merece esa calificación. No tiene barra espaciadora, posee una sola tecla SHIFT, RETURN está en una posición poco convencional y el interlineado no es exactamente igual al de una máquina de escribir

Conector para RF
Salida compatible para televisor; no dispone de salida para monitor

Conector para fuente de energía eléctrica
Aquí se aplica la energía proveniente de un pequeño transformador

RAM
Estos chips contienen los 4 K de memoria incorporados para el usuario

ROM
Estos chips retienen el BASIC Microsoft estándar de 8 K. El resto del espacio de la ROM está ocupado por las ampliaciones que se han agregado para manipular los gráficos y el sonido

Modulador
La señal de visualización en pantalla se convierte en una señal de TV estándar y aparece por el canal 36



Conector para impresora
Una interface para impresora exclusivo diseñado por Mattel se conecta a través de este enchufe, que es apto sólo para las dos impresoras que suministra la firma fabricante

Bus de ampliación
Aquí se pueden conectar numerosos accesorios, desde un módulo de RAM simple de 4 K hasta un chasis de ampliación, que puede recibir varios paquetes de RAM de 16 K así como una selección de programas útiles en paquetes de ROM

CPU
El procesador es un Z80, que funciona a una frecuencia de reloj de 3,5 MHz

Controlador CRT
El diseño de la electrónica que controla la visualización de video es el aspecto más importante del diseño informático. Este chip controlador es más grande que el propio microprocesador

Chip de seguridad
Este chip de diseño a medida está pensado para hacer muy difícil que alguien, excepto el fabricante del ordenador, pueda producir cartuchos de programas para ejecutar con el Aquarius

Conector para cinta
La interface para cinta es un enchufe tipo DIN y posee conexiones para controlar el motor de la grabadora de cassette

AQUARIUS

DIMENSIONES
345 × 150 × 55 mm

VELOCIDAD DEL RELOJ
3,5 MHz

MEMORIA
10 Kbytes de ROM, más 4 Kbytes de RAM, ampliables a 64 Kbytes

VISUALIZACION EN VIDEO
24 líneas de 40 caracteres, 16 colores con determinación de fondo y de primer plano independiente; 256 caracteres predefinidos, pero ningún carácter definible por el usuario

INTERFACES
Cassette, impresora, bus de ampliación

LENGUAJE SUMINISTRADO
BASIC Microsoft

OTROS LENGUAJES DISPONIBLES
Mattel tiene previsto incluir un BASIC Microsoft ampliado y un LOGO de Aquarius. Vendrán en forma de paquete de ROM

VIENE CON
Manual de instalación y manual de BASIC, cable para TV

TECLADO
49 teclas estilo botón. El mando de borrado (RESET) está protegido para evitar que se le pulse accidentalmente

DOCUMENTACION
La documentación es especialmente adecuada para principiantes, con un juego muy útil de tarjetas que describen cada una de las funciones principales de la máquina y del BASIC incorporado. Carece de detalles técnicos, pero, en general, es apropiada para el mercado hacia el que está destinado el Aquarius

Ramificación

A medida que se va desarrollando un programa, su estructura va tomando el aspecto de un árbol, que adquiere nuevas ramas al pasar cada una de las sucesivas etapas de refinamiento

En el capítulo anterior de nuestro curso de programación BASIC dimos una mirada a algunos de los problemas que implica la búsqueda a través de una lista para hallar un dato determinado (suponiendo que la lista ya estuviera clasificada por orden). Éste es un tema del que nos volveremos a ocupar con más detalle cuando llegue el momento de escribir rutinas de búsqueda. No obstante, mientras tanto desarrollaremos el tema de la programación *top-down* (de arriba abajo) para producir un código para las dos segundas partes del programa principal. Éste contiene cuatro llamadas a subrutinas o procedimientos:

```
PROGRAMA PRINCIPAL
EMPEZAR
   INICIALIZACION (procedimiento)
   PRESENTACION (procedimiento)
   ELECCION (procedimiento)
   EJECUCION (procedimiento)
FIN
```

El primer procedimiento, *INICIALIZACION*, implicará numerosas actividades bastante complicadas (establecer matrices, leer datos de ellas, realizar diversas verificaciones, etc.) y los detalles de su desarrollo los dejaremos para más adelante. Las dos partes siguientes del programa principal son las relativas a los procedimientos PRESENTACION y ELECCION. Para el desarrollo de estos procedimientos sugeriremos una metodología que ayude a evitar que se desorganicen y se confundan los muchos estratos involucrados en la realización de un programa *top-down*.

El problema que plantea el enfoque de un refinamiento de arriba abajo en el desarrollo de un programa, es que no se puede precisar el número de pasos necesarios antes de que estemos preparados para empezar la codificación en un lenguaje de alto nivel. Para los sistemas simples quizá podrían ser suficientes dos o tres pasos, pero los procedimientos más difíciles pueden requerir muchos antes de que el problema se haya analizado suficientemente como para permitir que se escriba el *código fuente* (así se denomina al programa en lenguaje de alto nivel). Esto significa que escribir un programa utilizando este método equivale a dibujar un árbol. A medida que van proliferando las "ramas" (es decir, a medida que los refinamientos se van volviendo más detallados), éstas van ocupando más lugar en la hoja. Finalmente, resulta imposible acomodar todo en una sola hoja y es en este punto donde resulta muy fácil perder la pista de lo que está sucediendo.

Una forma muy eficaz de organizar la documentación del programa consiste en numerar sistemáticamente las etapas de su desarrollo. Hemos empleado números romanos para indicar el nivel de refinamiento y números arábigos para señalar la subsección del programa. Después se utiliza una hoja separada de papel para cada uno de los niveles de refinamiento y las páginas para cada bloque o módulo de programa se pueden mantener juntas fácilmente. He aquí el sistema de numeración para nuestro programa:

```
I PROGRAMA PRINCIPAL
EMPEZAR
   1. INICIALIZACION
   2. PRESENTACION
   3. ELECCION
   4. EJECUCION
FIN
```

Como hemos mencionado más arriba, de momento estamos dejando de lado el desarrollo de INICIALIZACION, concentrándonos en desarrollar los procedimientos PRESENTACION y ELECCION.

```
II 2 (PRESENTACION)
EMPEZAR
   1. Visualizar mensaje de presentación
   2. LOOP (hasta que se pulse barra espaciadora)
      ENDLOOP
   3. Llamar *ELECCION*
FIN
```

```
III 2 (PRESENTACION) 1 (visualizar mensaje)
EMPEZAR
   1. Limpiar pantalla
   2. PRINT mensaje de presentación
FIN
```

```
III 2 (PRESENTACION) 2 (LOOP esperar barra
   espaciadora)
EMPEZAR
   1. LOOP (hasta que se pulse barra espaciadora)
         IF se pulsa barra espaciadora
         THEN
      ENDLOOP
FIN
```

```
III 2 (PRESENTACION) 3 (llamar *ELECCION*)
EMPEZAR
   1. GOSUB *ELECCION*
FIN
```

En este punto debería estar claro que **III-2-1** y **III-2-3** están listos para ser codificados directamente en BASIC, pero que **III-2-2** requiere otra etapa de refinamiento:

```
IV 2 (PRESENTACION) 2 (LOOP)
EMPEZAR
   1. LOOP (hasta que se pulse barra espaciadora)
         IF INKEY$ no es espacio THEN continuar
      ENDLOOP
FIN
```

Nos encontramos ahora en el punto donde con muy poco refinamiento mejor se puede abordar toda la codificación en BASIC para el procedimiento PRESENTACION:

IV 2 (PRESENTACION) 1 (visualizar mensaje) CODIGO BASIC

```
REM SUBRUTINA *PRESENTACION*
PRINT
PRINT
PRINT
PRINT
PRINT TAB(11);"*BIEN VENIDO A LA*"
PRINT TAB(9);"*AGENDA COMPUTERIZADA*"
PRINT TAB(12);"*DE MI COMPUTER*"
PRINT
PRINT TAB(0);"(PULSE BARRA ESPACIADORA PARA
   CONTINUAR)"
```

V 2 (PRESENTACION) 2 (LOOP esperar barra espaciadora) CODIGO BASIC

```
LET L = 0
FOR L = 1 TO 1
IF INKEY$ <>" " THEN LENT L = 0
NEXT L
```

IV 2 (PRESENTACION) 3 (llamar *ELECCION*) CODIGO BASIC

```
GOSUB *ELECCION*
RETURN
```

Observe que ahora hemos empezado a inicializar variables en las diversas rutinas que escribimos, utilizando sentencias en forma de LET I = 0. En rigor, esto no es necesario en algunas de las circunstancias en las que las hemos utilizado. No obstante, sería conveniente que usted se acostumbrara a ellas si puede recordarlas y si dispone de suficiente espacio de RAM. Y ello se debe a tres razones: primero, porque tener una lista de sentencias LET al comienzo de cualquier rutina sirve como un recordatorio útil de las variables locales que utiliza esa rutina. Segundo, porque puede que no esté seguro de lo que quedó en una variable desde la última vez que se la utilizó en una rutina (aunque esto no siempre es importante). Tercero, tal como le explicaremos cuando esté más avanzado el curso, porque colocar sentencias en forma de LET I = 0 en el orden correcto puede acelerar la ejecución de un programa.

Hemos alterado la forma en que utilizamos el bucle FOR...NEXT para simular una estructura DO...WHILE o REPEAT...UNTIL, explicadas en anteriores capítulos del curso. En vez de emplear FOR I = 0 TO 1 o FOR I = 0 TO 1 STEP 0, ahora utilizamos FOR I = 1 TO 1. Esto funcionará correctamente en todos los ordenadores personales con los que tratamos habitualmente, mientras que los otros procedimientos harían necesario "Complementos al BASIC" para varias máquinas. FOR I = 1 TO 1...NEXT I ejecutará el bucle sólo una vez. Sin embargo, si en algún punto del cuerpo del bucle I se estableciera en 0, entonces el bucle se ejecutaría otra vez, y así sucesivamente. Podemos insertar una sentencia LET I = 0 como resultado del fracaso de una condición de salida, o bien establecer I en 0 inmediatamente después de la sentencia FOR, y establecerlo en 1 si tuviera éxito la condición de salida. De modo, entonces, que los dos bucles siguientes alcanzan el mismo objetivo:

```
FOR I = 1 TO 1
IF INKEY$ <> " " THEN LET I = 0
NEXT I
```

o

```
FOR I = 1 TO 1
LET I = 0
IF INKEY$ = " " THEN LET I = 1
NEXT I
```

El código BASIC que acabamos de producir es todo cuanto se necesita para el bloque de PRESENTACION completo del programa principal. No hemos colocado números de línea porque realmente no podemos hacerlo hasta que todos los módulos del programa estén listos para la codificación final. Por ejemplo, en esta etapa no sabemos cuáles son los números de línea adecuados para las órdenes GOSUB. Si usted deseara comprobar el módulo en esta etapa, sería necesario crear algunas entradas ficticias y subrutinas ficticias. Algunos puntos de este fragmento de programa que es necesario señalar son el empleo de la función TAB y las sentencias para "limpiar la pantalla". TAB hace que el cursor se mueva a lo largo de la línea según el número (el "argumento") especificado entre paréntesis. Los números que hemos dado harán que el mensaje se imprima exactamente en el centro de una pantalla de 40 caracteres. Si su visualización fuera menos ancha que ésta (por ejemplo, el Spectrum visualiza 32 caracteres por línea) o más ancha (los ordenadores mayores normalmente visualizan 80 caracteres), será necesario modificar, consecuentemente, estos argumentos TAB. En muchas versiones de BASIC la instrucción para limpiar la pantalla es CLS, pero la versión de BASIC Microsoft utilizada para desarrollar este programa no la admite. En cambio, hemos empleado PRINT CHR$(12), dado que nuestra máquina utiliza ASCII 12 como su carácter no imprimible para "limpiar la pantalla" (otras suelen utilizar ASCII 24 para realizar la misma función).

```
10 REM PROGRAMA PRINCIPAL FICTICIO
20 PRINT CHR$(12)
30 GOSUB 100
40 END
100 REM SUBRUTINA *PRESENTACION*
110 PRINT
120 PRINT
130 PRINT
140 PRINT
150 PRINT TAB(11);"*BIEN VENIDO A LA*"
160 PRINT TAB(9);"*AGENDA COMPUTERIZADA*"
170 PRINT TAB(12);"*DE MI COMPUTER*"
180 PRINT
190 PRINT TAB(0);"(PULSE BARRA ESPACIADORA
    PARA CONTINUAR)"
195 LET L = 0
200 FOR L = 1 TO 1
210 IF INKEY$ <> " " THEN LET L = 0
220 NEXT L
230 PRINT CHR$(12)
240 GOSUB 1000
250 RETURN
1000 REM SUBRUTINA FICTICIA
1010 PRINT "SUBRUTINA FICTICIA"
1020 RETURN
```

Ahora utilizaremos exactamente el mismo enfoque para refinar el procedimiento ELECCION.

II 3 (ELECCION)
EMPEZAR
1. PRINT menú
2. INPUT OPCION
3. Llamar subrutina seleccionada
FIN

III 3 (ELECCION) 1 (PRINT menú)
EMPEZAR
1. Limpiar pantalla
2. PRINT menú y aviso
FIN

III 3 (ELECCION) 2 (INPUT OPCION)
EMPEZAR
1. INPUT OPCION
2. Verificar que OPCION esté dentro de la escala
FIN

III 3 (ELECCION) 3 (llamar OPCION)
EMPEZAR
1. CASO DE OPCION
FIN DEL CASO
FIN

Ahora **III-3-1 (PRINT menú)** se puede codificar en BASIC:

IV 3 (ELECCION) 1 (PRINT menú) CODIGO BASIC

```
REM LIMPIAR PANTALLA
PRINT CHR$(12):REM 0 'CLS'
PRINT
PRINT
PRINT
PRINT
PRINT "1.HALLAR REGISTRO (DE NOMBRE)"
PRINT "2.HALLAR NOMBRES (DE NOMBRE
        INCOMPLETO)"
PRINT "3.HALLAR REGISTRO (DE CIUDAD)"
PRINT "4.HALLAR REGISTRO (DE INICIALES)"
PRINT "5.LISTAR TODOS LOS REGISTROS"
PRINT "6.AGREGAR REGISTRO NUEVO"
PRINT "7.MODIFICAR REGISTRO"
PRINT "8.BORRAR REGISTRO"
PRINT "9.SALIDA Y GUARDAR"
```

Sin embargo, **III-3-2 (INPUT OPCION)** y **III-3-3 (llamar OPCION)** requieren todavía más refinamiento. Analicemos primero el siguiente nivel de desarrollo de **III-3-2**.

Asignarle un valor numérico a la variable OPCION es muy sencillo: después del aviso, lo hará una orden de INPUT OPCION. No obstante, sólo hay nueve opciones posibles. ¿Qué sucedería si por error diéramos entrada a 0 o 99? Dado que la OPCION que hagamos determinará a cuál de las partes del programa se llamará a continuación, deseamos asegurarnos de que no se produzcan errores, de modo que necesitamos llevar a cabo un procedimiento de "verificación de escala". Éste consiste en una pequeña rutina de verificación para ver si el número al que se ha dado entrada se halla dentro de la escala aceptada, antes de permitir que el programa continúe. He aquí una rutina de muestra diseñada para interrumpir una entrada errónea.

RUTINA DE VERIFICACION DE ESCALA

```
1 REM RUTINA
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT "DE ENTRADA A 1-9";OPCION
40 IF OPCION <1 THEN LET L = 0
50 IF OPCION >9 THEN LET L = 0
60 NEXT L
70 PRINT "LA OPCION ES";OPCION
80 END
```

Muchas versiones de BASIC pueden simplificar esta rutina mediante la inclusión en la condición de un operador de Boole como éste:

```
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT "DE ENTRADA A 1-9";OPCION
40 IF OPCION <1 OR OPCION>9 THEN LET L = 0
50 NEXT L
60 PRINT "LA OPCION ES";OPCION
70 END
```

Estas rutinas ilustran asimismo otro punto acerca de la sentencia INPUT. Esta sentencia hace que el programa se detenga y espere una entrada desde el teclado. El BASIC no sabe cuál es el número completo al que se ha dado entrada hasta que se pulsa la tecla RETURN, de modo que el usuario también tendrá que acordarse de pulsar RETURN después de dar entrada al número.

Un enfoque más "amable para el usuario" sería hacer que el programa continuara apenas se diera entrada a un número válido. Esto es posible gracias a la utilización de la función INKEY$. En este caso, el BASIC lee un carácter del teclado cada vez que se encuentra con INKEY$. Sin embargo, el programa no se detiene y continuará sin ninguna pausa por la parte siguiente. En consecuencia, es frecuente emplear INKEY$ dentro de un bucle. El bucle para comprobar si se está pulsando una tecla puede ser IF INKEY$ = "" THEN...; en otras palabras, si no se está pulsando ninguna tecla, vuelva y verifíquelo de nuevo. A nuestros fines, un bucle adecuado sería:

```
LET I = 0
FOR I = 1 TO 1
LET A$ = INKEY$
IF A$ = "" THEN LET I = 0
NEXT I
```

El único inconveniente que comporta la utilización de INKEY$ es que devuelve un carácter del teclado en vez de uno numérico. Cuando hay un menú de ELECCION, en el que se realiza una selección entre varias opciones (una ramificación multicondicionada), en BASIC es más fácil utilizar números que caracteres. Aquí es donde entran en juego las funciones NUM o VAL de BASIC. Éstas convierten a los números de las series de caracteres en números "reales" (es decir, valores numéricos y no códigos ASCII que representen numerales). Se pueden utilizar de la siguiente manera:

```
LET N = VAL(A$) o LET N = NUM(A$)
```

Utilizando las funciones NUM o VAL podemos hacer que, empleando INKEY$, el programa convierta las entradas en variables numéricas. Este procedimiento elimina la necesidad de emplear la tecla RETURN después de haber pulsado la tecla numérica. No obstante, es recomendable la verificación fuera de escala.

El siguiente fragmento de programa incluye dos bucles, uno anidado dentro del otro. El bucle inte-

rior espera a que se pulse una tecla; el bucle exterior convierte la variable en un número y verifica que esté dentro de la escala:

```
FOR L = 1 TO 1
   PRINT "DE ENTRADA A OPCION (1-9)"
      FOR I = 1 TO 1
         LET A$ = INKEY$
         IF A$ = "" THEN LET I = 0
      NEXT I
   LET OPCION = VAL(A$)
      IF OPCION <1 THEN LET L = 0
      IF OPCION >9 THEN LET L = 0
NEXT L
```

Por último, reproducimos un programa completo en BASIC para el módulo *ELECCION*, incluyendo subrutinas y entradas ficticias con fines de prueba. Debemos señalar, nuevamente, que los números de línea sólo se han incluido como prueba y habrán de ser sustituidos cuando se elabore el programa final.

```
10 PRINT CHR$(12)
20 PRINT "SELECCIONE UNA DE LAS
   SIGUIENTES OPCIONES"
30 PRINT
40 PRINT
50 PRINT
60 PRINT "1. HALLAR REGISTRO (DE NOMBRE)"
70 PRINT "2. HALLAR NOMBRES (DE NOMBRE
   INCOMPLETO)"
80 PRINT "3. HALLAR REGISTRO (DE CIUDAD)"
90 PRINT "4. HALLAR REGISTRO (DE INICIALES)"
100 PRINT "5. LISTAR TODOS LOS REGISTROS"
110 PRINT "6. AGREGAR REGISTRO NUEVO"
120 PRINT "7. MODIFICAR REGISTRO"
130 PRINT "8. BORRAR REGISTRO"
140 PRINT "9. SALIDA Y GUARDAR"
150 PRINT
160 PRINT
170 LET L = 0
180 LET I = 0
190 FOR L = 1 TO 1
200 PRINT "DE ENTRADA A OPCION (1-9)"
210 FOR I = 1 TO 1
220 LET A$ = INKEY$
230 IF A$ = "" THEN LET I = 0
240 NEXT I
250 LET OPCION = VAL(A$)
260 IF OPCION <1 THEN LET L = 0
270 IF OPCION >9 THEN LET L = 0
280 NEXT L
290 ON OPCION GOSUB 310,330,350,370,390,410,
    430,450,470
300 END
310 PRINT "SUBRUTINA FICTICIA 1"
320 RETURN
330 PRINT "SUBRUTINA FICTICIA 2"
340 RETURN
350 PRINT "SUBRUTINA FICTICIA 3"
360 RETURN
370 PRINT "SUBRUTINA FICTICIA 4"
380 RETURN
390 PRINT "SUBRUTINA FICTICIA 5"
400 RETURN
410 PRINT "SUBRUTINA FICTICIA 6"
420 RETURN
430 PRINT "SUBRUTINA FICTICIA 7"
440 RETURN
450 PRINT "SUBRUTINA FICTICIA 8"
460 RETURN
470 PRINT "SUBRUTINA FICTICIA 9"
480 RETURN
```

En el próximo capítulo analizaremos las estructuras de archivo y empezaremos a refinar el procedimiento INICIALIZACION.

Complementos al BASIC



En el programa principal ficticio, y de principio a fin, sustituir PRINT CHR$(12) por CLS y END por STOP.

RUTINA DE VERIFICACION DE ESCALA

```
1 REM RUTINA
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT "ENTRADA A 1-9";OPCION
40 IF OPCION <1 THEN LET L = 0
50 IF OPCION >9 THEN LET L = 0
60 NEXT L
70 PRINT "LA OPCION ERA";OPCION
80 STOP
```

LISTADO FINAL

```
10 CLS
```

después copiar la lista del texto principal hasta:

```
240 NEXT I
250 LET OPCION = CODE A$ - 48
260 IF OPCION <1 THEN LET L = 0
270 IF OPCION >9 THEN LET L = 0
280 NEXT L
290 GOSUB (OPCION*20 + 290)
300 STOP
```

luego copiar la lista principal desde la línea 310 hasta la 480.



Algunas versiones del Oric-1 no obedecen a la orden TAB, aun cuando está incluida en el BASIC del Oric-1; en este caso, insertar esta línea al principio del programa:

```
5 LET S$ = ""
```

En esta línea, entre las comillas debería haber tantos espacios como caracteres haya en una línea de pantalla completa (40, para un Oric-1). Después, siempre que el programa diga TAB(11), reemplácelo por LEFT$(S$,11), copiando el número de la sentencia TAB en la función LEFT$( ).



En el Oric-1, el Dragon 32, el Lynx y el BBC Micro, sustituir PRINT CHR$(12) por CLS. En el Commodore 64 y en el Vic-20, para reemplazar CHR$(12) consulte el manual.



No está disponible en el Lynx, pero se puede sustituir por la línea 290 del listado final que hemos dado arriba para el Spectrum



Ver "Complementos al BASIC", p. 257.



Ver "Complementos al BASIC", p. 175. Los usuarios de un Commodore han de sustituir LET A$ = INKEY$ por GET A$, e IF INKEY$ = "" THEN por:
GET A$:IF A$ = "" THEN

Tony Lodge

Un útil "ratón"

Los diseñadores desean reemplazar el teclado por un dispositivo de uso más sencillo. Éste podría ser el "ratón"

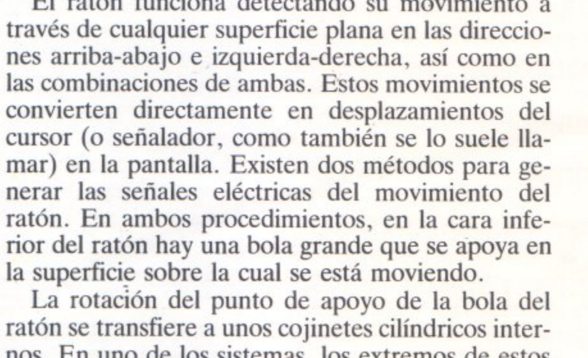
Hasta no hace mucho, la única forma de acceder a los ordenadores era a través de unas enormes máquinas de escribir electromecánicas llamadas "teletipos". Éstos eran dispositivos ruidosos, difíciles de manejar y nada fiables, que desde entonces han ido siendo sustituidos por la silenciosa y veloz VDU (*Visual Display Unit:* unidad de representación visual) con teclado. La VDU eliminó muchos de los problemas relacionados con los teletipos, uno de los cuales era la enorme cantidad de papel consumido en las cintas perforadas a medida que se iba digitando la información. No obstante, tanto el terminal mecánico como la VDU más teclado están limitados por su formato carácter a carácter, línea a línea. El usuario no puede desplazarse rápidamente por la pantalla (seleccionar ítems de un menú por aquí, modificar un dato por allá, o cambiar archivos y programas) sin encontrarse con las limitaciones del formato del cursor tecleado. La liberación del teclado se consigue al emplear terminales para gráficos o al practicar juegos por ordenador con mandos de bola y palancas de mando; pero ¿qué utilidad pueden reportar estos dispositivos si se quiere destinar el ordenador a otros fines?

La mayoría de los ordenadores personales que existen en el mercado están equipados con mandos para el cursor en cuatro direcciones, que se pueden desplazar a través de un listado de programa o del texto de un documento hasta la posición donde se necesita hacer una corrección. Pero el cursor sólo se puede mover por pasos de una línea o un carácter; el usuario no lo puede desplazar directamente hasta su destino. Si el cursor de texto fuera susceptible de moverse como un cursor de gráficos, que se puede manipular con entera libertad bajo el control de un mando de bola o de una palanca de mando, los datos se podrían desplazar de una manera considerablemente más rápida.

En el Stanford Research Institute de California se estudió por primera vez una solución a este problema, en los años sesenta; y el primer "ratón" (tal como se bautizó al nuevo tipo de controlador que se desarrolló) se patentó en 1970. Al dispositivo se lo denominó *ratón* debido a su aspecto: un ratón es lo suficientemente pequeño como para caber en la palma de la mano; tiene un "rabo" (el cable), y los primeros dispositivos solían tener dos "orejas" (los botones de control). No se utilizan mandos de bola ni palancas de mando convencionales porque no se requiere la precisión que éstos proporcionan para colocar el cursor en la posición deseada.

El ratón funciona detectando su movimiento a través de cualquier superficie plana en las direcciones arriba-abajo e izquierda-derecha, así como en las combinaciones de ambas. Estos movimientos se convierten directamente en desplazamientos del cursor (o señalador, como también se lo suele llamar) en la pantalla. Existen dos métodos para generar las señales eléctricas del movimiento del ratón. En ambos procedimientos, en la cara inferior del ratón hay una bola grande que se apoya en la superficie sobre la cual se está moviendo.

La rotación del punto de apoyo de la bola del ratón se transfiere a unos cojinetes cilíndricos internos. En uno de los sistemas, los extremos de estos cilindros están provistos de ruedas de código que poseen pistas alternadas de material conductor y no conductor. Los impulsos recibidos los cuenta el



"Tres ratones ciegos"
Muchos de los microordenadores para gestión empresarial más recientes incorporan un ratón como estándar, y algunas empresas ofrecen unidades como accesorios para máquinas ya existentes. La mayoría de los ratones opera mediente una bola rotatoria e incorpora uno, dos o tres botones de "SELECT"



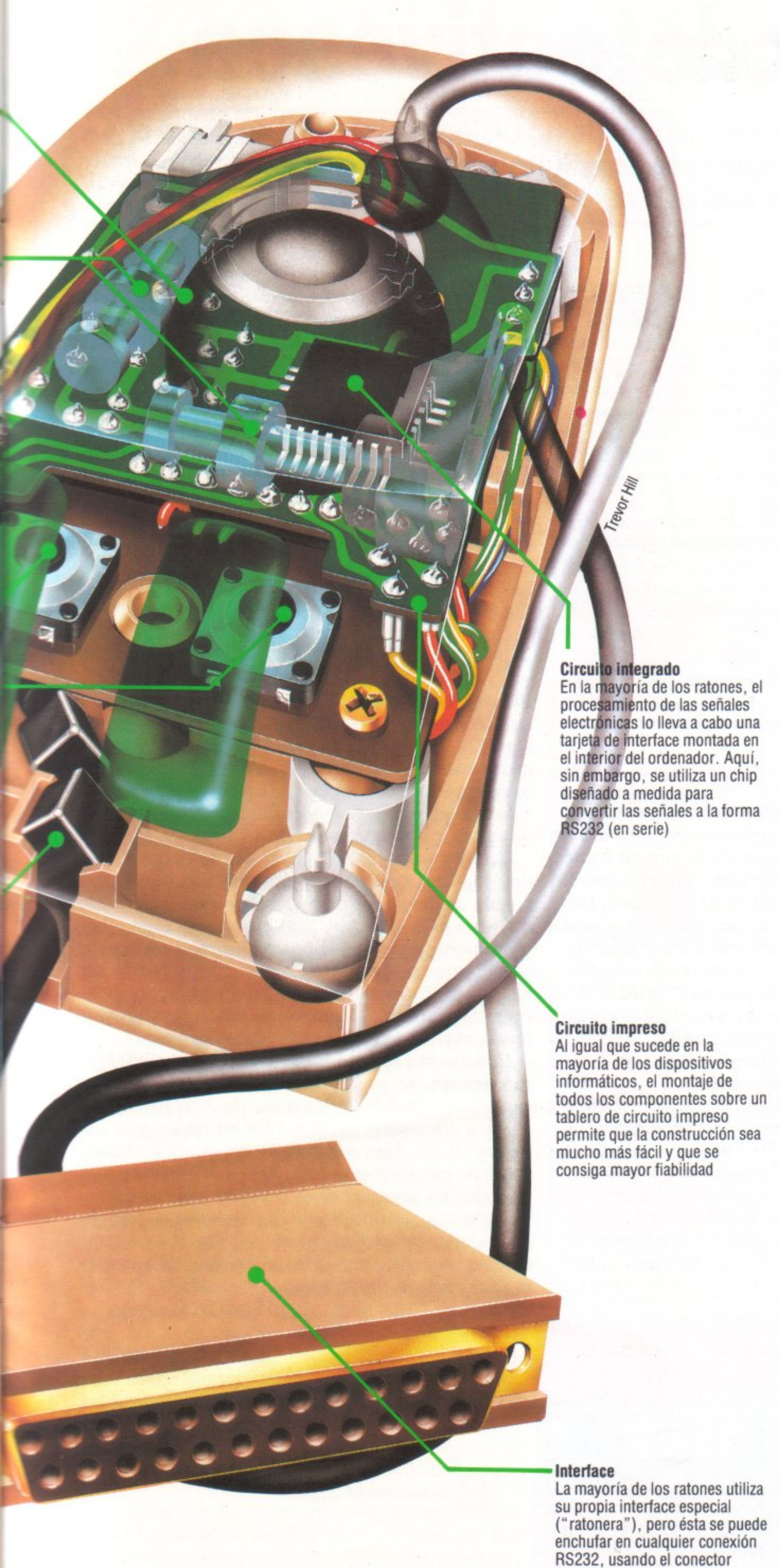
Bola principal
Una gran bola de apoyo de acero descansa sobre la superficie a través de la cual se mueve el ratón. La bola de algunos ratones es de plástico duro para evitar que patine

Ruedas de codificación
Estas dos ruedas hacen contacto constante con la bola para captar su movimiento en dos direcciones. Las ruedas están montadas sobre ejes; al extremo de estos ejes hay dispositivos de codificación que a medida que giran aquéllos producen impulsos eléctricos

Botones
La función de los dos botones depende del paquete de software que se utilice. Por lo general, uno se emplea para seleccionar un ítem y el otro para mover objetos a través de la pantalla

Microinterruptores
Éstos están montados en el circuito impreso por debajo de los botones, y sólo requieren un mínimo movimiento para crear o romper el circuito

Anillo de goma
El ratón ha de tener libertad para moverse por el escritorio, y el anillo de goma es particularmente importante para evitar la tensión en la conexión entre el cable y el circuito impreso



Circuito integrado
En la mayoría de los ratones, el procesamiento de las señales electrónicas lo lleva a cabo una tarjeta de interface montada en el interior del ordenador. Aquí, sin embargo, se utiliza un chip diseñado a medida para convertir las señales a la forma RS232 (en serie)

Circuito impreso
Al igual que sucede en la mayoría de los dispositivos informáticos, el montaje de todos los componentes sobre un tablero de circuito impreso permite que la construcción sea mucho más fácil y que se consiga mayor fiabilidad

Interface
La mayoría de los ratones utiliza su propia interface especial ("ratonera"), pero ésta se puede enchufar en cualquier conexión RS232, usando el conector estándar de 25 canales

software operativo del ratón y le permiten dar una lectura de la posición del cursor en la pantalla. En el otro sistema, los cojinetes llevan acoplados dos discos acanalados. A los discos se les dirige continuamente una luz y, al otro lado de ellos, una célula fotoeléctrica detecta ópticamente el haz. Los impulsos de luz que pasan a través de las ranuras se convierten luego en señales eléctricas, que se tratan de la misma manera que en el sistema mecánico.

Existen, asimismo, otros sistemas. Hay uno, por ejemplo, en que el ratón se utiliza en conjunción con un relleno especial cubierto por un patrón de puntos. Una luz en el interior del cuerpo del ratón ilumina la superficie de relleno cubierta por el ratón y este patrón lo detecta un chip procesador óptico especial. Cualquier movimiento del ratón hará que cambie el patrón que detecta el chip, que puede calcular al instante hasta dónde se ha movido el dispositivo y en qué dirección. Este sistema ofrece la ventaja de que no posee partes móviles, pero es mucho más caro que los otros.

Una vez que se ha desplazado el cursor hasta el lugar de la pantalla requerido, se puede dar entrada a su posición en el ordenador pulsando una de las "orejas" (botones) del ratón. El número de botones de que dispone el ratón varía de un fabricante a otro. Algunos sistemas utilizan tres; Microsoft ha optado por incorporarle dos, mientras que el ratón del Lisa de Apple posee sólo uno. Los botones también se pueden emplear para seleccionar ítems de un menú (programas como el *MultiTool Word* de Microsoft ofrecen esta facilidad) y para confiarle al ratón el control del movimiento del cursor normal. Estos dispositivos se pueden utilizar con software altamente sofisticado, como el que se proporciona con el Lisa. Aquí el botón se pulsa una vez para seleccionar un "icono" (véase p. 262) de un menú en pantalla, y dos veces para posibilitar esa aplicación determinada.

La ventaja principal de todos los ratones, y del software que se ha producido para complementarlos, es que los pueden utilizar todas aquellas personas que no poseen experiencia con teclado. En vez de tener que digitar el nombre de un programa o de pulsar ciertas letras o números para seleccionar una función, el usuario simplemente mueve el ratón de modo que el cursor de la pantalla señale la aplicación o el cursor de acción que se requiera, y pulsa un botón para activarlo.

Lamentablemente, este nuevo dispositivo no elimina por completo la necesidad del teclado (aún se debe alimentar el ordenador con textos y números nuevos), pero simplifica en gran medida la manipulación de la información. Las pruebas que llevó a cabo la Apple durante el desarrollo del Lisa demostraron que un usuario que no tuviera ninguna experiencia con un ordenador podía aprender a trabajar con el software activado por ratón del Lisa en apenas 15 minutos. Ejecutado en un sistema convencional, familiarizarse con un software similar lleva aproximadamente 20 horas, básicamente debido a los problemas que entraña aprender a emplear el teclado y a la necesidad de aprender órdenes largas y complicadas. Los ratones electrónicos muy pronto serán un componente integral de los ordenadores personales. Son eficaces y fáciles de utilizar, y a las personas poco emprendedoras no las atemorizan tanto como la simple vista de un teclado QWERTY tradicional.

Labor detectivesca

Cuando se pasa información de un ordenador a otro, se corre el riesgo de que ciertos datos se alteren. Los códigos Hamming pueden detectar y corregir estos errores

Todos hemos oído alguna historia acerca de garrafales errores cometidos por ordenadores, como, por ejemplo, enviarle por correo 500 ejemplares del folleto de una empresa a una misma persona. La verdad, por supuesto, es que la máquina no tiene culpa alguna: la equivocación se origina en un fallo humano, quizá tan sencillo como un error de digitación. El ordenador sirve tan sólo para magnificar el problema.

Algunas veces, no obstante, los ordenadores cometen fallos no imputables a la intervención humana que por lo general se manifiestan en forma de "errores de bits". Un error de bits se produce cuando se transpone un bit simple de una sección de datos de 1 a 0 o viceversa. Un error de bits puede surgir cuando falla un componente del hardware, como un chip de RAM. Por ese motivo muchos ordenadores personales se someten a un software de "diagnóstico" para verificación de errores cada vez que se encienden.

Sin embargo, la mayoría de los errores de bits son "errores de *soft*": los bits "se dan vuelta" aun cuando toda la RAM haya pasado la prueba de diagnóstico. Los ordenadores personales están diseñados para trabajar en interiores, pero durante una intensa ola de calor en verano, es muy posible que la temperatura supere la escala térmica operativa de los componentes. Es poco probable que el daño sea de tipo permanente, pero los errores de bits podrían ocasionar que un carácter de la pantalla cambiara súbitamente de una "A" a una "B", por ejemplo, o, en el caso de que el bit formara parte de un indicador importante, podría "romper" el programa, haciendo necesario restaurarlo.

Los errores de bits también pueden surgir en períodos de intensa actividad de las manchas solares, cuando partículas subatómicas pueden penetrar en la atmósfera e interferir el flujo de electrones de un circuito en miniatura. En aplicaciones tales como sistemas militares, control industrial, experimentos científicos o movimiento bancario internacional, los errores pueden tener consecuencias desastrosas, de modo que para detectarlos se han adoptado diversos procedimientos.

El más sencillo de estos métodos es el del control de paridad (véase p. 253). Un procedimiento alternativo es la suma de control, que se utiliza mucho al escribir datos en cinta magnética o en disco. Comúnmente, los datos se manipulan en bloques de 128 bytes, de los cuales el último en leerse o escribirse será el byte de suma de control. Éste representa la suma de los otros bytes (cada uno de los cuales posee un valor entre 0 y 255) módulo 256; su significado corresponde al resto de la suma cuando se la ha dividido por 256. He aquí un ejemplo:

Datos: 114,67,83... (otros 121 valores)...
36,154,198
Total de estos 127 bytes = 16 673
Total dividido por 256 = 65, resto 33
Por tanto, suma de control = 33

El total de los bytes (16 673) es igual a 65 porciones de 256 más un resto de 33 (el valor escrito en el byte 128 como suma de control). Cuando el ordenador vuelve a leer el bloque, efectúa su propio cálculo de suma de control de la información, y si este valor difiere de 33, entonces sabe que en el proceso de grabación se ha producido un error de bits.

Tanto al emplear el método de paridad como el de suma de control, el ordenador no dispone de ningún medio que le permita saber cuál es el bit del dato que se ha alterado. Si el error se produjo en la transmisión, entonces el ordenador receptor puede solicitar que se le vuelva a transmitir un byte o un bloque de bytes determinados; en el caso de un error de grabación, bien podría ser que no hubiera forma alguna de recuperar el dato correcto.

Cuando sea indispensable evitar que se produzcan errores, se ha de utilizar un sistema que sea capaz tanto de detectarlos como de corregirlos. Los *códigos Hamming*, así llamados en honor a su inventor, R. W. Hamming, de Bell Telephone Laboratories, cumplen esta función.

Todos los sistemas de corrección de errores trabajan sobre el principio de la redundancia. El lenguaje humano contiene un elevado nivel de redundancia; si al mecanografiar un manuscrito se comete un error, o si durante una conversación telefóni-

A	0	0	1	1
B	0	1	0	1
R	0	1	1	0

Puerta "Or" exclusiva
Una puerta "Or" exclusiva sencilla tiene dos entradas y una salida. Si ambas entradas están en un 0 lógico, entonces la salida es 0. Si alguna de las entradas es 1, entonces la salida es 1. No obstante, si ambas entradas son 1, entonces la salida es 0. Esta última situación es la que determina la diferencia entre la puerta Or y la Or-ex (para mayor brevedad). La operación se puede representar con una tabla de verdad. Cuando una Or-ex posee más de dos entradas, la salida será 1 si en la entrada hay un número impar de unos. Es mediante dispositivos de este tipo como se crean los bits de control de paridad y de error

Kevin Jones

ca alguna interferencia impide oír algunas palabras, con frecuencia éstas se pueden deducir del contexto de la oración. A veces incorporamos redundancia extra al hablar en entornos "ruidosos": cuando en las comunicaciones telefónicas empleamos los términos "Almería", "Barcelona" y "Cáceres" para dar a entender inequívocamente que nos referimos a una "a", una "b" y una "c", por ejemplo.

Supongamos que desde un ordenador enviamos una palabra de x bits de longitud, compuesta por y bits de datos reales y por z bits redundantes (es decir, $x = y + z$). En nuestra explicación de la paridad teníamos para y un valor de siete y para z, otro de uno. Para los códigos Hamming, z habrá de ser proporcionalmente mayor. Supongamos ahora que en cualquiera de los x bits se pueda producir un error de un solo bit (nuestros bits redundantes z, por supuesto, son tan susceptibles de error como los bits de datos y). Si la probabilidad de que en una palabra se produzca un error de bits es, pongamos por caso, de una en un millón, la probabilidad de que en una palabra tengan lugar dos errores es de una en un millón de millones, de modo que descartaremos esta eventualidad.

Cuando los datos se reciban al otro extremo, habrá $x + 1$ probabilidades. O no habrá ningún error, o habrá un error en el primer bit de datos, y así sucesivamente hasta el último bit x. Ahora bien, con bits redundantes z podemos representar 2^z situaciones, de modo que para que la palabra esté a prueba de un error de bits:

$$2^z \geq y + z + 1$$

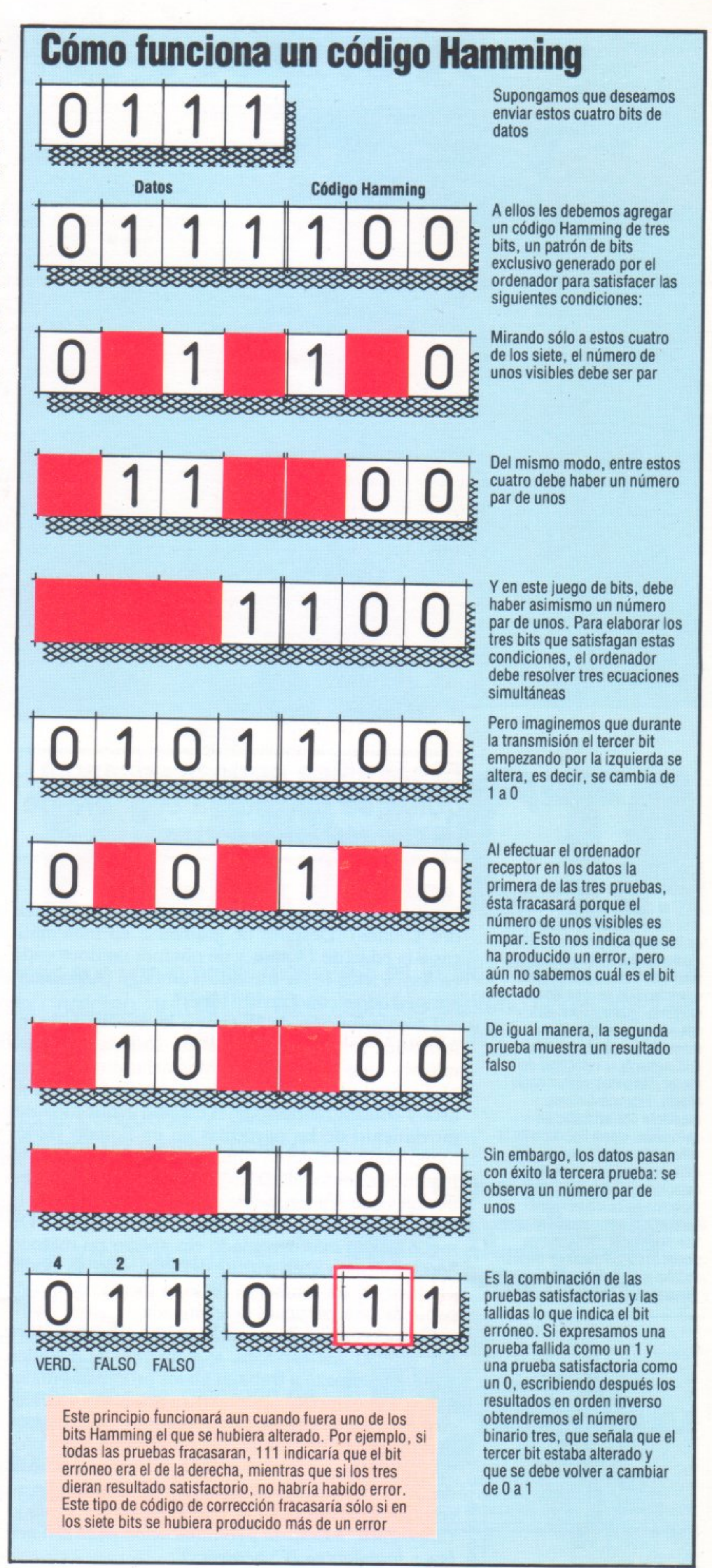
Si y es siete (en código ASCII), entonces z habrá de ser cuatro. Si y es cuatro (como en nuestro ejemplo del panel), z habrá de ser tres. Sin embargo, si y es 16, z sólo se habrá de aumentar a 5. Los códigos Hamming son más eficaces para palabras largas que para las cortas.

En un código Hamming, cada uno de los bits redundantes actúa como un control de paridad par en una combinación diferente de los bits de la palanca. Si en la transmisión se cambiara algún bit, uno o más de los bits de control estaría mal y la combinación de estos bits señalaría el bit erróneo de la palabra (véase ejemplo). El software del ordenador receptor puede entonces volver a cambiar el bit.

La clave para la forma en que funcionan los códigos Hamming son las distintas combinaciones de bits sobre las cuales actúa como control de paridad cada bit Hamming. El número total de bits, efectivamente, se divide en juegos distintos pero superpuestos, diseñados para que no aparezcan dos bits en la misma combinación de juegos. El ordenador receptor efectúa los controles de paridad en los mismos juegos que lo hizo el dispositivo transmisor para crear el código Hamming. Si cualquiera de los bits, incluyendo los bits Hamming, se hubiera alterado durante la transmisión, entonces uno o más de estos juegos no pasaría la prueba de paridad.

Algunos ordenadores utilizan los códigos Hamming incluso para sus operaciones de memoria interna. Cuando es éste el caso, ¡se puede quitar un chip de RAM entero y ver cómo el ordenador sigue funcionando! Algunos ordenadores destinados a uso militar llevan el principio de la redundancia hasta el extremo de duplicar cada componente simple del ordenador y comparar, luego, los resultados de las dos mitades después de cada operación.

Cómo funciona un código Hamming



Norbert Wiener



El científico norteamericano a quien se considera el padre de la ciencia cibernética

Norbert Wiener nació en 1894 en Missouri (Estados Unidos). Después de graduarse en matemáticas a la edad de 14 años y de obtener un doctorado en lógica a los 18, se trasladó a Gotinga (Alemania) para estudiar con David Hilbert.

La contribución de Wiener a la ciencia de la informática se produjo al final de su vida. Durante muchos años trabajó en el Massachusetts Institute of Technology, estudiando la nueva física probabilista y concentrándose en el estudio estadístico del movimiento de las partículas en un líquido (fenómeno conocido como "movimiento browniano"). Los movimientos de las partículas eran tan impredecibles que era imposible describirlos utilizando la física tradicional de las fuerzas deterministas. De modo que lo más apropiado era aplicar un método "probabilístico", en virtud del cual sólo se podía predecir en un momento dado la localización *probable* de una partícula determinada.

Cuando estalló la segunda guerra mundial, Wiener ofreció sus servicios al gobierno de Estados Unidos y empezó a trabajar en los problemas matemáticos que implica apuntar un arma hacia un blanco móvil. El desarrollo de los sistemas automáticos para el guiado de la mira, sus estudios de física probabilística y su marcado interés por temas que iban desde la filosofía hasta la neurología, todo ello se conjugó en 1948 cuando publicó un libro titulado *Cibernética, o Control y comunicación entre el hombre y la máquina (Cybernetics)*.

La cibernética es el estudio de los controles autogobernados que existen en los sistemas estables, ya sean mecánicos, eléctricos o biológicos. Fue Wiener quien vio que la información era cuantitativamente tan importante como la energía o la materia: un alambre de cobre, por ejemplo, se puede estudiar por la energía que puede transmitir o la información que puede comunicar. La revolución que anuncia el ordenador se basa en parte en esta idea: la fuente de poder pasa de la propiedad de la tierra, la industria o la empresa al control de la información. Su contribución a la ciencia de la informática no consistió en el diseño de elementos de hardware, sino en la creación de un medio intelectual en el cual se pudieron desarrollar los ordenadores y los autómatas.

El término *cibernética* deriva de una palabra griega que significa "arte de gobernar". Wiener había estudiado el regulador del motor a vapor de James Watt, que ajustaba automáticamente la velocidad de la máquina, y comprendió que para que fuera posible desarrollar los ordenadores, éstos deberían imitar la capacidad de los seres humanos de regular sus propias actividades.

El termostato de una casa constituye un ejemplo de un sistema de control. Regula la calefacción según las fluctuaciones de la temperatura por encima o por debajo de un nivel óptimo. Sólo se necesita que un ser humano determine este nivel. A esta facultad de autorregulación y control Wiener la denominó *feedback negativo* ("feedback", porque la salida del sistema —el calor— afecta al comportamiento futuro del sistema, y "negativo", porque las variaciones del termostato se producen para restaurar la temperatura al nivel establecido).

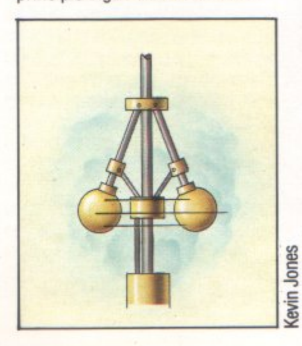
Se dice que el sistema que puede cumplir esta función y también seleccionar su propia temperatura (y lograr otros objetivos) es un sistema de *feedback positivo*. Cuando un autómata puede realizar todos estos cometidos y además reproducirse a sí mismo, entonces se acerca a la condición humana.

La teoría de la cibernética de Wiener se puede considerar como una superciencia (una ciencia de ciencias) y ha fomentado la investigación en muchas áreas de sistemas de control y de sistemas que tratan con la información. Todo es información. Todo cuanto sabemos acerca de los cambios del mundo nos llega a través de nuestros ojos y nuestros oídos y otros receptores sensoriales, que son dispositivos para seleccionar sólo ciertos datos de un total que, de lo contrario, nos desbordaría.

La información también se puede estudiar de forma estadística, independientemente de cualquier significado que pueda tener. Por ejemplo, observando la frecuencia con que se producen ciertos símbolos se pueden interrumpir muchos tipos de códigos. En castellano, las letras "a" y "e" están entre las que aparecen con mayor frecuencia. Analizando grandes muestras de un código y comparando los resultados con muestras típicas de castellano, se pueden identificar letras clave y, por consiguiente, empezar a descifrar el código.

Wiener murió en 1964, antes de que empezara la revolución del microordenador, a pesar de lo cual previó muchos de los problemas que surgirían en esta nueva tecnología y escribió acerca de ellos.

Restricción de velocidad
Wiener quedó fascinado con la idea del regulador a vapor, uno de los mejores y más sencillos ejemplos de feedback negativo. Mediante brazos pivotantes se conectan dos pesas a un eje giratorio, que a su vez está conectado al volante de la máquina de vapor. A medida que aumenta la velocidad del motor, las pesas vuelan hacia afuera. Este movimiento, mediante una articulación apropiada, cierra ligeramente la válvula de estrangulación del motor. Esto tiene el efecto de establecer la velocidad de la máquina en cualquier nivel determinado por el operador. Los ordenadores modernos pueden realizar tipos de control mucho más sofisticados, pero el principio sigue siendo el mismo



Kevin Jones

UNION PERFECTA

Así se comportan los periféricos creados por SINCLAIR para SINCLAIR: de forma perfecta. Y es lógico.

Cada vez que SINCLAIR diseña un microordenador, no lo hace de una manera aislada. Simultáneamente crea todos esos periféricos que van a hacer más potente, preciso y útil el microordenador que tiene entre manos.

Periféricos pensados y diseñados para dar un servicio óptimo, pero con un precio razonable, dentro de la filosofía SINCLAIR:

"Hacer la informática accesible a todos".

Por eso cuando creó el ZX 81 vio la necesidad de dotarlo con una ampliación de memoria de 16K RAM para que no quedara pequeño y de una impresora sencilla y barata pero útil y precisa.

Así es la filosofía SINCLAIR. Así son los periféricos de SINCLAIR para SINCLAIR.





CENTRAL COMERCIAL: Tomás Bretón, 60
Tel. 468 03 00 Telex: 23399 IYCO E Madrid.
DELEGACION CATALUÑA: Camp, 80 - Barcelona - 22

PARA JUGAR A LO GRANDE
(INSTANTANEAMENTE)

Presentamos **el Interface 2 ZX** Pensado y diseñado por SINCLAIR para unirse a la perfección con tu microordenador Spectrum.

Si a la hora de elegir tu microordenador optaste por el mejor, es lógico que elijas ahora el Interface 2 ZX

Ya habrás podido deleitarte con la más amplia variedad de juegos existentes para tu Spectrum (la más extensa del mercado). Ahora con el Interface 2 ZX vas a tener más ventajas para tu Spectrum:

- Podrás conectar Joysticks para sacarle, aún, mayor rendimiento a tus mejores juegos y divertirte con aquellos exclusivamente disponibles en **Cartuchos ZX:** correr, saltar, volar... a lo grande. ¡Menuda diferencia!
- Además, al ser cartuchos con memoria ROM, podrás, con tu SPECTRUM de 16 K, jugar con programas hasta ahora reservados para 48 K, sin ampliar la memoria. ¡Vaya ahorro!
- Al conectar el Interface 2 ZX tienes la certeza de poseer un periférico pensado por SINCLAIR para SINCLAIR. Tu microordenador queda a salvo de circuitos poco fiables. ¡Un alivio!
- Al adquirir el Interface 2 ZX y los Cartuchos ZX en la red de Concesionarios Autorizados, podrás exigir la tarjeta de garantía INVESTRONICA, única válida en territorio nacional. ¡Una tranquilidad!



Interface 2 ZX y Cartuchos ZX

Si aún no los tienes
no sabes lo que te pierdes

Solicita una demostración en cualquier Concesionario Autorizado INVESTRONICA.



CENTRAL COMERCIAL: Tomás Bretón, 60
Tel. 468 03 00 Telex: 23399 IYCO E Madrid.
DELEGACION CATALUÑA: Camp, 80 - Barcelona - 22